US-PAT-NO: 6167438

DOCUMENT-IDENTIFIER: US 6167438 A

TITLE: Method and system for distributed caching, prefetching and replication

---------- KWIC ---------

Abstract Text - ABTX (1):
A technique for automatic, transparent, distributed, scalable and robust caching, prefetching, and replication in a computer network that request messages for a particular document follow paths from the **clients** to a home **server** that form a routing graph. **Client** request messages are routed up the graph towards the home **server** as would normally occur in the absence of caching. However, cache servers are located along the route, and may intercept requests if they can be serviced. In order to be able to service requests in this manner without departing from standard network protocols, the cache **server** needs to be able to insert a packet filter into the router associated with it, and needs also to proxy for the home **server** from the perspective of the **client**. Cache **servers** may cooperate to service **client** requests by caching and discarding documents based on its local load, the load on its neighboring caches, attached communication path load, and on document popularity. The cache servers can also implement security schemes and other document transformation features.

Application Filing Date - AD (1):
19970522

Brief Summary Text - BSTX (3):
Other computers in the network, known as **clients,** allow a user to access a document by requesting that a copy be sent by the home **server** over the network to the **client**. In order for a **client** to obtain information from a home **server,** each document typically has an address by which it can be referenced. For example, in the context of the Internet and within the communication protocol known as Hyper Text Transfer Protocol (HTTP), the address is typically an alphanumeric string, known as a Uniform Resource Locator (URL), that specifies (a) an address of the home **server** from which to obtain the information in the form of a name or a numerical address, and (b) a local information text string that identifies the information requested by the **client,** which may be a file name, a search request, or other identification.

Brief Summary Text - BSTX (4):
After the user specifies a URL to the **client computer, the address** portion of the URL is sent over the network to a naming service such as the Domain Name Service (DNS) in order to obtain instructions for how to establish a connection

with the correct home **server**. Once the connection with the **server** is established, the **client** can then retrieve the desired document by passing the local information text string over the network directly to the home **server**. The **server** then retrieves the document from its local disk or memory storage and transmits the document over the network to the **client**. The network connection between the home **server and the client** is then terminated.

Brief Summary Text - BSTX (6):
The present bottlenecks are no doubt the result of exponential increases in the number of users as well as in the number of complex documents such as multimedia files being sent. It might appear that the answer is simply to add more bandwidth to the physical connections between **servers and clients**. This will come, however, only at the expense of installing high bandwidth interconnection hardware, such as coaxial or fiber optic cable and associated modems and the like, into homes and neighborhoods around the world.

Brief Summary Text - BSTX (8):
An approach taken by some has been to recognize that the rapidly growing use of the Internet will continue to outstrip server capacity as well as the bandwidth capacity of the communication media. These schemes begin with the premise that the basic **client-server** model (where **clients** connect directly to home **servers**) is wasteful of resources, especially for information which needs to be distributed widely from a single home **server** to many **clients**. There are indeed, many examples of where Internet servers have simply failed because of their inability to cope with the unexpected demand placed upon them.

Brief Summary Text - BSTX (9):
To alleviate the demand on home servers, large central document caches may be used. Caches are an attempt to reduce the waste of repeated requests for the same document from many **clients** to a particular home **server**. By intercepting parallel requests, a cache can be used to serve copies of the same document to multiple client locations.

Brief Summary Text - BSTX (11):
Alternatively, a **server** node, acting as a proxy for the **client**, may issue probe messages to search for a cache copy. Once a cache copy is found at a particular node in the network, the request is then forwarded to that node. For example, under the auspices of the National Science Foundation, document caches have been placed at various locations in the United States in order to eliminate bottlenecks at cross-oceanic network connections. Generally, certain of these caches located on the West Coast handle requests for documents from the Asia-Pacific and South American countries, and a number of those located on the East Coast handle requests for documents from Europe. Other of these national caches handle requests for popular documents located throughout the United States.

Brief Summary Text - BSTX (14):
Another possible approach to implementing caches is to change the

client/server interaction protocol so that **clients** proactively identify suitable cache copies using a fully distributed protocol, for example, by issuing probes in randomized directions. Aside from the complexity of modifying existing protocols and message cost introduced by such an approach, such a scheme also adds one or more round trip delays to the total document service latency perceived by the client.

Brief Summary Text - BSTX (16):
The present invention is an automatic, distributed, and transparent caching scheme that exploits the fact that the paths that document requests follow through a computer network from a **client** to a particular document on a particular home **server** naturally form a routing graph, or tree.

Brief Summary Text - BSTX (17):
According to the invention, cache **servers** are placed throughout the network, such that if a document request can be fulfilled at some intermediate node along the routing graph, it will be serviced by the intermediate node returning the cached document to the **client**. The document request messages are thus responded to before they ever reach the home server. Since document request messages are permitted to be routed from **clients** in the direction of the home **server** up the routing graph in the same manner as would occur in the absence of caching, naming services do not need modification.

Brief Summary Text - BSTX (19):
The cache server also preferably acts as a communication protocol proxy for the home server. That is, as part of fulfilling document request messages at the intermediate node locations, the **client** is sent appropriate messages, depending upon the communication protocol in use, to spoof the **client** into believing that the document was actually received from the home **server**.

Brief Summary Text - BSTX (20):
The invention also provides a manner in which caching **servers** may cooperate to service **client** requests. In particular, each server has the ability to cache and discard documents based on its local load, the load on its neighboring caches, adjacent communication path load, and on document popularity. For example, each server maintains an estimate of the load at its neighbors, and communicates its own load estimate to neighboring cache servers. If a cache server notices that it is overloaded with respect to any of its neighbors, it offloads or transfers a fraction of its work to its under loaded neighbors. To do so, a cache server also preferably learns the identity of its neighboring upstream (or parent)and downstream (or child) nodes on the routing graph that is rooted at a given home server.

Brief Summary Text - BSTX (22):
First, the approach does not need to request an **address lookup from a cache directory, to redirect** document requests, or to otherwise probe other elements of the network to locate cache copies. Location of the cache copy thus occurs fortuitously, along the natural path that the request message follows anyway.

The client thus does not experience delays or bottlenecks associated with waiting for other entities in the network to find appropriate cache copies.

Brief Summary Text - BSTX (24):
There is also a corresponding reduction in network bandwidth consumption and **response time,** because cache copies are always placed nearer to the original **server than to the client,** Document request messages and the documents themselves therefore typically do not need to travel the full distance between the **server and each client** every time they are requested. Hence, overall network bandwidth is conserved, **response times** are reduced, and load is more globally **balanced**.

Brief Summary Text - BSTX (27):
The technique is also scalable, in the sense that as more cache **servers** are added, both **clients and servers** experience a likewise benefit.

Brief Summary Text - BSTX (32):
One such feature is the authentication of the sources of request messages and other information. This is possible because the cache servers maintain information as to the physical source of document request messages and of the documents themselves. The mechanism also arises from the fact that the nodes have a filter and a packet router. The filter and packet router may be used not only to keep track of how to redirect requests to cache copies, but also to restrict access to the cache copies, such as by authenticating the request for the information. The invention also enables various types of document distribution in a the network. For example, the invention permits document compression, which is another form of conserving bandwidth, or encryption, as long as a particular **server and client** node are committed to communicating by using cache **servers,** e.g., the first and last nodes along the path between the **client and the server** in the network contain cache **servers**.

Brief Summary Text - BSTX (34):
The invention also improves the delivery of stored continuous media such as audio and video data files since the number of network nodes between a **server and a client** are reduced.

Detailed Description Text - DETX (5):
The **clients** 12 and home **server** 20 operate as in the prior art to permit distribution of a wide variety of information, typically in the form of "documents". Such documents may actually contain text, graphics, pictures, audio, video, computer programs and any number of types of information that can be stored in a computer file or parts of a computer file. Furthermore, certain documents may be produced at the time that access is requested to them, by executing a program.

Detailed Description Text - DETX (6):
It will be assumed in the following discussion that the network 10 is the

Internet, that the information is encoded in the form of the Hyper Text Transfer Protocol (HTTP) documents, and that document request messages are sent in the form of Uniform Resource Locators (URLs) using the TCP/IP layered protocol. This is with the understanding that other types of wired, switched, and wireless networks, and other types of protocols such as FTP, Gopher, SMTP, NNTP, etc. may make advantageous use of the invention. In addition, although the invention is discussed in the context of a **client-server** type of communication model, it should be understood that the principals of the invention are equally applicable to peer-to-peer networks.

Detailed Description Text - DETX (7):

A request message for a particular document, for example, originates at one of the client computers, such as client 12-1. The message is a request by the **client** 12 for the home **server** 20 to send a copy of document that is presently stored at the home **server** 20 location such as on a disk. The document request message is passed through one or more routers 14, such as routers 14-1, 14-2, 14-3, in the direction of the illustrated arrows, on its way to the home server 20.

Detailed Description Text - DETX (8):

In networks such as the Internet, document request messages may pass through as many as fifteen or more nodes or "hops" through routers 14 before reaching their intended destination. Requests for the same document from other **clients, such as clients** 12-2, 12-3, or 12-4 also pass through different routers 14 on their way to the home **server** 20 at the same time.

Detailed Description Text - DETX (10):

A model is useful for understanding the nature of how requests from multiple clients for one particular document travel across a path the computer network 10. The model is that structure, T, which is induced by the effect of routing algorithm on the document request messages as they travel through the network to the home server 20. As shown in FIG. 1, the home **server** 20 can thus be thought of as being at the root node of the structure, T, with document requests originating at the leaf node levels farthest away from the root, namely at **clients** 12-1, 12-2, . . . , 12-4. The structure T also includes many intermediate nodes which are located the routers 14.

Detailed Description Text - DETX (11):

While the structure T of the set of paths that **client** requests follow towards a given home **server** 20 is accurately and generally described as a data directed, acyclic graph, the present exposition does not benefit from the added complexity. In particular, when a single particular document is considered as being located at only one home server, the structure can be referred to as a tree with a single root. With that understanding we use the term tree to describe the structure T herein, with the understanding that a graph model may also be used. With this model in mind, the entire Internet can be thought of as a forest of trees or graphs, each rooted at a different home server 20 which is responsible for providing an authoritative permanent copy of some set of documents.

Detailed Description Text - DETX (12):
In accordance with the invention, copies of documents are located in the network at cache servers 16. According to the invention, the placement of cache copies, and hence the diffusion of load, is constrained to nodes in the tree structure, T. This avoids the need for **clients** to lookup the locations of cache copies, either by directly contacting the home **server** 20, or a naming service such as a Domain Name Service (DNS), or by probing the network in search of apropriate cache copies.

Detailed Description Text - DETX (13):
The present invention also assumes that cache **servers** 16 lie on the path along the tree that document request messages would naturally take from the **client** 12 to the home **server** 20, with the cache **servers** 16 cooperating to off-load excess load at the home **server** 20, or to diffuse other potential performance bottlenecks such as communication links themselves. In effect, the routers 14 having associated cache servers 16 inspect document request message packets as they fly-by and intercept any request for which it may be possible to fulfill by providing a cached document instead.

Detailed Description Text - DETX (14):
In a most general description of the operation of the invention, document request messages travel up the tree T, from a **client** at which it originated, such as **client** 12-3, towards the home **server** 20. Certain routers encountered by the document request message along the way, such as router 14-7, do not have local cache servers 16, and thus simply pass the document request message up to Findeathe next router in the tree, such as router 14-6.

Detailed Description Text - DETX (15):
However, certain other routers, such as router 14-6, do have a local cache server 16-6, in which case the document request message is examined to determine if it is seeking a document located in the local cache store 18. If a cache copy is encountered at cache **server** 16-6, then that copy is returned to the **client** 12, and the request message is not permitted to continue on its way to the home **server** 20. If however, a cache copy is not encountered at the particular cache server 16-6, the request message continues to the next router 14-4 on the path to the home server 20.

Detailed Description Text - DETX (17):
Ideally, the implementation of the cache **servers** 16 is such that no changes are required to the normal operating mode of either **clients** 12 or **servers** 20. Another goal is to have a design that can be gradually deployed into the existing infrastructure of the network 10. This also requires that any new mechanisms preferably be compatible with existing communication protocols.

Detailed Description Text - DETX (22):
To accomplish this load management, or load **balancing,** the resource manager

24 maintains information about the identity and the load of its neighboring cache servers 30. The details of how neighboring cache server information is maintained is discussed below in Section 3.

Detailed Description Text - DETX (24):

Other responsibilities of the resource manager 24 include neighborhood discovery, propagating load information to the neighboring servers 30, and discovering and recovering from potential barriers to load **balancing**. These mechanisms are discussed in more detail below.

Detailed Description Text - DETX (25):

The routers 14 take an active role in assisting cache servers 16 to achieve cache server and/or communication path **balancing** goals. This is accomplished by allowing the resource manager 24 to inject functionality into the router 14 in the form of the code that implements the filter 26 and snooper 28. In particular, all packets passing through a router 14 not addressed directly to a host server 20 are first passed to the snooper 28. The snooper 28 inspects a packet and determines its type, destination, and the document requested. Depending on the state of the cache server 16 and packet type, the snooper 28 could intercept the packet or simply forward the packet to the next hop, or router 14, along the intended destination path to the home server 20.

Detailed Description Text - DETX (28):

2. Handling an HTTP Document Request in a TCP/IP Network Current implementations of networks 10 that use HTTP rely on the layered TCP/IP protocol for reliable end-to-end communication between **clients** 12 and **servers** 20. This layering divides the normal processing of a request message into three steps; connection establishment (i.e., TCP-level three way handshake in the form of [SYN] messages), HTTP document request/reply in the form of [GET] messages, and connection termination in the form of [FIN] messages.

Detailed Description Text - DETX (29):

This process is depicted in FIG. 3, where the **client** 12 first issues a [SYN] message with a sequence number to the home **server** 20, and the home **server** 20 returns a [SYN] message with an acknowledgment [ACK]. In response to this, the client 12 then sends a document request in the form of a [GET] message that includes the URL of the desired document. The document is then forwarded by the home **server** 20 to the **client** 12. After the **client** 12 returns an acknowledgment, the **server** 20 and **client** 12 terminate the connection by exchanging [FIN] and [ACK] messages.

Detailed Description Text - DETX (30):

The main hurdle in actually implementing the cache **servers** 16 as explained above in such an environment is the requirement that they need to identify the document requested by a **client** 12. However, as seen in FIG. 3 the URL information is typically advertised by an HTTP **client** 12 only after a TCP/IP connection has already been established with the home **server** 20. One possible solution would thus be to have all such connections be established with the

home server 20 and have snoopers 28 at intermediate routers 14 intercept all [GET] packets. Even though this approach might relieve a significant amount of load from a home server, it still required that TCP connections associated with such documents reach the home server 20, which defeats the purpose of attempting to off-load the home server 20. During high demand periods, such requests would amount to a flood of [SYN] requests on the home server 20. In addition, if the initial [SYN] is not intercepted, both establishing and tear down of connections becomes significantly more complicated.

Detailed Description Text - DETX (32):
This functionality is implemented by the snooper 28. In particular, snoopers 28 located in routers 14 on the path to a home server 20 inspect packets that fly-by, identify such packets, and intercept any [SYN] packets directed to HTTP home servers 20. As [SYN] packets do not contain any information identifying which document the **client** 12 intends to request, the snooper 28 acts as a proxy for, or "spoofs" the home **server** 20, by establishing a connection between the **client** 12 and the local transport layer in the cache **server** 16, and noting the initial sequence numbers used by both the **client** 12 and the local transport layer.

Detailed Description Text - DETX (34):
In a first approach, the TCP connection is handed off, wherein the snooper 28 closes the **server** half of the spoofed TCP connection with the **client** 12, and forwards the document request in the form of a composite "piggy back" [SYN+GET] message in the direction of the home **server** 20. In addition, the [SYN+GET] message contains all the state information needed to hand-off the server half of the TCP connection to any other intermediate cache server on the path to the home server 20 which happens to cache the requested document.

Detailed Description Text - DETX (35):
In a second alternative approach, the snooper may act as a TCP relay, maintaining the TCP connection with the **client,** and relaying the [SYN+GET] message on a separate connection to the next intermediate cache **server** on the path to the home **server** 20.

Detailed Description Text - DETX (36):
The above hand-off process is illustrated in the flow chart of FIG. 4. This process is carried out by a particular class of cache **servers** 16 referred to as leaf node **servers** 38, which are the cache **servers** 16 that are on the extreme lower level nodes of the tree T, i.e., the first **servers** to intercept a [SYN] packet from a **client** 12. The leaf node servers 28 in the tree T depicted in FIG. 1 are cache servers 16-1, 16-6, and 16-8.

Detailed Description Text - DETX (37):
As shown in step 41 of FIG. 4, when a leaf node **server** 38 receives a [SYN] packet, the home **server** 20 is proxied for, or "spoofed", by establishing a TCP connection directly between the leaf node **server** 38 and the **client** 12. The leaf node **server** 38 then waits to intercept the corresponding [GET] request

from the **client** 12.

Detailed Description Text - DETX (38):
Note that spoofing thus occurs in the sense that packets exchanged between the **client** 12 and a cache **server** 16 are modified by the snooper 28 in the above scenario. In particular, the network address of a cache server 16 which is servicing a request is replaced with the network address of the home server 20 and in a connection hand-off, the sequence numbers of bytes issued by the cache server 16 have to follow the sequence number as determined by the leaf server 38.

Detailed Description Text - DETX (39):
Returning to step 41, if the requested document passes the cache query test by the filter 28, and in step 42, and if the resource manager 22 detects that the document is present in the local cache and will permit access to it, then the document request is serviced locally, in step 45. In step 45, the [GET] command is forwarded to the resource manager, which then replies with the requested document. Finally, the TCP connection between the leaf **server** 38 and the **client** 12 is closed, by spoofing the home **server** 20 once again and issuing the closing [FIN] and [ACK] messages to the **client**.

Detailed Description Text - DETX (43):
If the request can be processed locally, step 55 completes the proxying for the home **server** 20 by establishing the **server** half of the TCP connection with the **client** 12, issuing the [GET] to the resource manager 24, returning the document to the **client** 12, and closing the TCP connection.

Detailed Description Text - DETX (48):
However, if the next node is a home server 20, then the step 63 is performed. In particular, snooper 28 establishes the **server** half of the TCP connection with the **client** 12, and replaces the [SYN+GET] with a [PROXY.sub.-- GET] request to the local resource manager 24. The resource manager 24 translates the [PROXY.sub.-- GET] request to an explicit [GET] issued to the home server 20. The response of the home **server** 20 response is then relayed to the **client** 12 in the same manner as if the cache **server** was caching the requested document.

Detailed Description Text - DETX (49):
Another shortcoming of the caching technique described thus far is that the path along the tree T between a particular **client** 12 and the home **server** 20 can change after a leaf node **server** 38 or an intermediate node **server** 39 decides to service a request. This may occur, for example, when a network connection, or link, is lost between two server nodes. FIG. 7 shows this relatively rare case where the path between the **client** 12 and the home **server** 20 changes while an intermediate cache **server** 16b is processing a document request from **client** 12. All [ACK]s sent by the **client** 12 will now follow the new path, through a new cache **server** 16x, to the home **server** 20. This causes cache server 16b to time-out and retransmit its packets.

Detailed Description Text - DETX (50):
To solve this problem, the snooper 28 at server 16b may keep track of the number of times a packet is re-transmitted. If a packet is re-transmitted more than a predetermined number of times, for example, three times, the snooper 28 then assumes that the path between the **client** 12 and the home **server** 20 has changed, and then takes steps to terminate the connection with the **client** 12. In particular, the snooper 28 aborts the connection with the **client** 12 and aborts the connection with cache **server** 16b, simultaneously spoofing the home **server** 20 and sending a reset packet (i.e., an [RST] packet) to the **client** 12.

Detailed Description Text - DETX (51):
In another approach the leaf node **servers** 28 closest to the **clients** 12 and the last hop nodes closest to the **server** 20 are provided with only one possible route to the **clients** 12 and **servers** 20, respectively. This is accomplished by having the cache **servers** forward **client** request messages over cache **server**-to-cache **server** permanent TCP connections, instead of simply letting the request messages follow their normal routes. The set of connections, being implemented as a set of properly joined TCP connections, thus automatically adapts to any changes in IP routing as the network configuration changes.

Detailed Description Text - DETX (53):
However, any resulting changes in the configuration of adjacent cache servers must also be detected by communication with neighboring cache servers in order to achieve resource load **balancing** and other advantages possible with the invention. In particular, each cache server 16 participating in the above-described scheme has to determine which other servers are in its neighborhood. In addition, on each routing tree T, a cache server 16 has to distinguish between upstream servers (located at parent nodes) and down stream servers (located at child nodes). A particular node, i, in the tree T is the parent of a node j, if i is the first cache server 16 on the route from j to the home server 20, in which case node j is also referred to as the child of node i.

Detailed Description Text - DETX (59):
As shown in FIG. 8, a typical HTTP [GET] message follows a path from the **client** 12 through A to the home **server** 20 and is intercepted by intermediate cache 16c. While cache **server** 16c is processing the request, the path between the home **server** 20 and the **client** 12 changes causing all acknowledgments to use a different path.

Detailed Description Text - DETX (64):
4. Load **Balancing**

Detailed Description Text - DETX (65):
Unlike most other caching schemes, the caching scheme according to the invention requires distribution of cache copies to the cache **servers** 16 prior

to **clients** actually requesting them. In other words, documents are moved among the cache **servers** 16 in anticipation of future document requests, rather than in direct response to any one particular document request message by the **clients** 12.

Detailed Description Text - DETX (66):
The above scheme of document caching and neighborhood discovery lends itself to a number of different types of such cache load distribution and/or load **balancing** objectives for both the cache servers 16 as well as the communication paths which interconnect them. In the preferred embodiment, this load distribution scheme attempts to avoid introducing an overhead that grows quickly with the size of the caching system, by using a diffusion based caching algorithm that relies strictly on local information.

Detailed Description Text - DETX (67):
In particular, the resource managers 24 create cache copies only when an upstream ("parent") node in the routing tree T detects a less loaded downstream ("child") node or link, to which it can shift some of its document service load by giving it a copy of one of its cached documents. "Load" herein can be a number of different performance criteria such as rate of request fulfillment, **client response time,** or fraction of time the **server** is busy.

Detailed Description Text - DETX (72):
A first goal is to determine how a cache server selects a document to pass to an underloaded neighbor. An objective here is to extract the maximum capacity of all of the cache servers 16 in the network. A second objective is to reduce **response time,** by moving popular documents closer to clients and less popular documents away from clients, all by creating the least number of replicas. These goals are accomplished while also considering communication path load between cache servers 16.

Detailed Description Text - DETX (84):
With large documents, it may be advantageous for the local cache 18 to only include a portion of the requested document. In this event, the cache **server** can begin to provide the locally cached portion of the document to the **client** 12, while at the same time requesting the remainder of the document be sent by the home **server** 20.

Detailed Description Text - DETX (102):
The cache servers may also implement authentication of the sources of request messages and other information. This can be done because the cache **servers** 16 automatically maintain information as to the **client** 12 which was the source of a particular document request message. If the **client** 12 is not among the authorized requesters for the document, the request message can be terminated at the cache **server** 16 before it even reaches the home **server** 20.

Detailed Description Text - DETX (103):

Selective security can also be provided for as well. The mechanism arises from the fact that the nodes each have a filter 26, a resource manager 24, a cache repository 18, and an HTTP proxy 22. The filter 26 may be used not only to keep track of how to redirect requests for cache copies to the HTTP proxy 22, but may also restrict access to the cache copies, such as by authenticating the request for the information. As long as the first and last hop along the path from the **client** 12 to the **server** 20 are trusted, since the links between the caches **servers** 16 can easily be arranged to be trusted links, a secure link can be provided between the **clients** 12 and the home **server** 20 via the cache **servers** 16.

Detailed Description Text - DETX (104):
More generically, the cache **servers** 16 may transform documents at several points during their distribution by the home **server** 20 and subsequent delivery to **clients** 12. Such transformations may enable and/or implement several features that add value to caching a document or program, including such features as compression or encryption of documents and/or programs. Furthermore, any known security techniques can be applied as transformations at the source, destination or both (in the case of a transaction). In addition to encryption and decryption, these include authentication, authorization (access control), non-repudiation, and integrity control. These security techniques can use cryptographic techniques that usually require a key, and optionally use the physical path to a cache server to strengthen authentication, authorization, and non-repudiation.

Detailed Description Text - DETX (106):
FIG. 10 illustrates one possible approach for applying transformations to documents. In this example, distinct transformations may occur at four different times as a document is distributed by the home **server** 20 and delivered to the **client** 12:

Detailed Description Text - DETX (111):
Note that even though the transformations in FIG. 10 are associated with four phases of communication, in reality, transformations are performed by the nodes themselves. Thus, T1 may be performed on the request by cache **server** 16-6 after it is received, or by **client** 12 before it is sent. In the former case, cache server 16-6 is performing the transformation. In the later case, the **client** 12 is performing the transformation, and merely tunneling this through cache **server** 16-6. Likewise, as the request is forwarded to the home server 20 the transformation T2 is performed by cache server 16-7 or home server 20. The same alternatives apply for T3 and T4, as the reply is sent from home **server** 20 and ultimately forwarded to **client** 12.

Detailed Description Text - DETX (112):
Where the transformations as performed in FIG. 10 have an important role is in key placement for security (or any other transformations which requires input other than the document). If the cache servers 16 themselves are implementing security, cryptographic keys must be distributed to the cache servers 16, as needed. In this case, end-to-end security can be provided by

adding a secure channel between **client** 12 and cache **server** 16-6 as well as between cache **server** 16-7 and **server** home 20. If the **client** 12 and home **server** 20 implement their own end-to-end security the cache **servers** 16 do not hinder, and possibly cooperate in, distributing keys to the **client** 12 and home **server** 20.

Detailed Description Text - DETX (113):
FIG. 10 shows the cache **servers** 16-6 and 16-7 at the edges of the network, whereby there is a direct path between **client** 12 and cache **server** 16-6 as well as between cache **server** 16-7 and home **server** 20. However, the design as described does not preclude a more general configuration, when transformations are performed by intermediate cache servers 16-9 and 16-10. In other words, in FIG. 10 a transformation may be made at an intermediate cache server 16-9, 16-10 during the request, or reply, or both.

Detailed Description Text - DETX (114):
The scheme also allows for a form of transparent document compression, which is another form of conserving bandwidth, as long as a particular home **server** 20 and **client** 12 are committed to communicating by using cache **servers** 16. In particular, after the first hop along the path between the **client** 12 and the **server** 20, the first leaf node **server** 16-6 can implement compression or encryption in a manner which is transparent to both the **client** 12 and the home **server** 20.

Detailed Description Text - DETX (118):
If, however, the programs are of the type which the **client** 12 expects will be running on the home **server** 20, the cache **server** 16 performs an additional level of home **server** spoofing by also running the programs. In this instance, it may be necessary for the cache **server** 16 to maintain an interactive session state with the **client** 12 in order to complete the spoofing of the home **server** 20.

Detailed Description Text - DETX (119):
The invention also inherently improves the delivery of stored media such as audio and video data files since number of hops between a home **server** 20 and a **client** 12 are reduced.

Claims Text - CLTX (1):
1. In a system containing a plurality of computers which communicate over a network using communication protocols, with the computers at certain nodes in the network acting as home **servers** for storing information in the form of documents, and with certain other computers acting as **clients** that send document request messages to the **servers** at an application layer, the document requests message being requests for documents stored at the home **servers,** a method of fulfilling document request messages comprising the steps of:

Claims Text - CLTX (3):

(b) in response to a particular one of the **clients** generating a particular application layer document request message addressed to a particular one of the home **servers,** attempting to fulfill the particular application layer document request message at one of the intermediate node locations by, at a selected communication layer lower than the application layer,

Claims Text - CLTX (4):
(i) intercepting the document request message at an intermediate node location through which the document request naturally travels, the intermediate node being located along a path through the network from the **client** towards the home **server,** the path being a route from the **client** to the home **server** as determined by the home **server** address, and corresponding to an identical path that the message would travel through the network in the absence of any cache copies being stored at intermediate node locations; and

Claims Text - CLTX (5):
(ii) returning one of the local cache copies to the application layer at the **client,** such that the application layer request message is intercepted by the lower layer at the intermediate node and such that the application layer on the **server** does not receive application layer document request message.

Claims Text - CLTX (8):
3. A method as in claim 1 wherein the step of fulfilling the particular document request additionally comprises, in at least one intermediate node, wherein the intermediate node is neither a **client** nor a home **server,** the step of:

Claims Text - CLTX (15):
(f) exchanging status messages between the intermediate node and the neighboring node, the status messages including information selected from at least one of processing load, communication path load, document size, document request message **response time,** document request message request rate, document request message rate of change, or home server operability.

Claims Text - CLTX (22):
(e) determining an identity of a second neighboring node to which a particular document request message is routed on the path to the home **server** if the particular document request message cannot be fulfilled by returning the local cache copy to the **client**.

Claims Text - CLTX (25):
10. A method as in claim 9 wherein the node state parameters are selected from the group consisting of rate of request fulfillment, change in rate of request fulfillment, document size, document fetch **response time,** communication path load, cache server load, or home server operational status.

Claims Text - CLTX (28):
12. A method as in claim 11 wherein the node state parameters are selected from the group consisting of rate of request fulfillment, change in rate of request fulfillment, document size, document fetch **resp nse time,** communication path load, cache server load, or home server operational status.

Claims Text - CLTX (38):
(c) recording **response time** statistics as to the number of request messages received for documents stored at a particular home server.

Claims Text - CLTX (51):
(d) maintaining an interactive session between the intermediate node and the **client** such that the intermediate node acts as the home **server** would act in the event that the home **server** had fulfilled the document request message.

Claims Text - CLTX (55):
27. A method as in claim 8 wherein multiple intermediate nodes are located between the home **server and the client,** and wherein a particular document is pushed into the network by routing it to a plurality of nodes depending upon an expected rate of demand upon the document.

Claims Text - CLTX (56):
28. A method as in claim 9 wherein the predetermined rate of request fulfillment depends upon document attributes selected from the group of expected rate of request fulfillment, expected change in rate of request fulfillment, document size, expected document fetch **response time,** expected communication path load, or expected cache server load.

Claims Text - CLTX (63):
33. A method as in claim 31 wherein the predetermined conditional criteria is at least one selected from the group of document size, desired document request message **response time,** document request message rate, rate of change of the document request message rate, cache server load, communication path load, or home server operational status.

Claims Text - CLTX (68):
37. A method as in claim 35 wherein the predetermined conditional criteria is at least one selected from the group of document size, desired document request message **response time,** document request message rate, rate of change of the document request message rate, cache server load, or communication path load.

Claims Text - CLTX (76):
40. In a system containing a plurality of computers which communicate over a network using a layered communication protocol, with the computers at certain nodes in the network acting as home **servers** for storing information in the form

of documents, and with certain other computers acting as **clients** that send document request messages to the **servers** at an application layer level, the document request messages being requests for documents stored at the home **servers,** a method of fulfilling document request messages by transparent proxying comprising the steps of:

Claims Text - CLTX (78):
(b) in response to a particular one of the **clients** generating a particular application layer document request message intended to be sent to a particular one of the home **servers,** fulfilling the particular application layer document request message at one of the intermediate node locations along a path from the **client** to the home **server** through which the document request would naturally flow in the absence of any cache **servers,** by intercepting the document request message and returning one of the local cache copies in a response message to the application layer at the **client,** such that the application layer request message is intercepted at the intermediate node and such that a network connection is not established with the application layer on the home **server** and such that the addresses in a response message are the same as if the home **server** had fulfilled the document request message.

Claims Text - CLTX (84):
(c) generating a connection request message which requests a communication path to be established between the **client** and a home **server**; and at an intermediate node,

Claims Text - CLTX (85):
(d) upon receiving a connection request message from the **client,** waiting for the receipt of a document request message, and forwarding the connection request message and the document request message together addressed to the home **server**.

Claims Text - CLTX (98):
49. A method as in claim 48 wherein the node state parameters are selected from the group consisting of rate of request fulfillment, change in rate of request fulfillment, document size, document fetch **response time,** communication path load, home server operability, or cache server load.

Claims Text - CLTX (99):
50. In a system containing a plurality of computers which communicate over a network, with certain computers acting as home **servers** for storing information in the form of documents, and with certain other computers acting as **clients** that send messages which are requests for documents stored at the home **servers,** a method of fulfilling document request messages by distributed caching comprising the steps of:

Claims Text - CLTX (100):
(a) routing document request messages from a **client** to a home **server** along a

path from the **client** to the home **server,** the path comprising a plurality of hops including a plurality of intermediate node locations in the network through which the document request would naturally flow through in the network in the absence of cache **s rvers**;

Claims Text - CLTX (105):
51. In a system containing a plurality of computers which communicate over a network, with the computers at certain nodes in the network acting as home **servers** for storing information in the form of documents, and with certain other computers acting as **clients** that send document request messages which are requests for documents stored at the home **servers,** a method of fulfilling document request messages at node through which the document request messages travel comprising the steps of:

Claims Text - CLTX (107):
(b) in response to a particular one of the **clients** generating a particular document request message intended to be sent to a particular one of the home **servers**;

Claims Text - CLTX (108):
(i) determining a path for the particular document request message to travel from the particular **client** to the particular home **server** along a path of nodes located in the network between the particular **client** and the particular home **server,** the path being determined by an address of the home **server** and the path comprising a plurality of the intermediate node locations in the network;

Claims Text - CLTX (121):
(c) if the leaf node does not store the requested document opening a communication connection between the leaf node and the next intermediate node by sending a connection request message addressed to the home **server** that is intercepted by the next intermediate node **server,** the leaf node being one of the intermediate node locations that initially receives the document request message from the **client**; and

Claims Text - CLTX (124):
(c) opening a communication connection between a leaf node and the home **server,** the leaf node being one of the intermediate node locations that initially receives the document request message from the **client**; and

Claims Text - CLTX (127):
(c) acting as a communication proxy for the home **server** from the perspective of the **client**.

Claims Text - CLTX (139):
(c) controlling access to the cache copy of the document based upon **client** authentication and home **server** request.

Claims Text - CLTX (140):

66. A method of fulfilling requests for information in a network of computers, with at least some of the computers in the network acting as home **servers** for storing information, and at least some of the computers acting as **clients** that send request messages to the home **servers,** the method comprising the steps of:

Claims Text - CLTX (142):

(b) routing request messages from the **client** to the home **server** along a path consisting of a plurality of intermediate computers in the network, the path including computers through which the request messages travel in the absence of any cache **servers** with at least some of the intermediate computers also acting as cache **servers,** the request messages initiated by a particular one of the **clients** to obtain information from a particular one of the home **servers**; and

Claims Text - CLTX (143):

(c) transparently processing request messages as they are routed through the cache **servers,** such that request messages that can serviced by the cache **servers** instead of the home **servers** are serviced by the cache **servers,** in a manner which is transparent to the **clients** and the home **servers**.

Other Reference Publication - OREF (2):

Heddaya, Abdelsalam and Mirdad, Sulaiman "WebWave: Globally Load **Balanced** Fully Distributed Caching of Hot Published Documents" Boston University, Computer Science Department, BU-CS-96-024, Abstract, (Oct. 15, 1996).

Other Reference Publication - OREF (12):

Heddaya, A., et al., "WebWave: Globally Load **Balanced** fully distributed Caching of Hot Published Documents," Technical Report BU-CS-96-024, Boston University, CS Department, (10/96), downloaded from Internet at: http://www.cs.bu.edu/techreports/abstracts/96-024.

Other Reference Publication - OREF (13):

Heddaya, A., et al., "WebWave: Globally Load **Balanced** Fully Distributed Caching of Hot Published Documents," Proceedings of the 17.sup.th International Conference on Distributed Computing Systems (Cat. No. 97CB36053), Proceedings of 17.sup.th International Conference on Distributed Computing Systems, Baltimore, MD, pp. 160-168 (May 27-30, 1997).

US-PAT-NO: 6665702

DOCUMENT-IDENTIFIER: US 6665702 B1

TITLE: Load **balancing**

---------- KWIC ---------

Application Filing Date - AD (1):
19991220

TITLE - TI (1):
Load **balancing**

Parent Case Text - PCTX (2):
This application is a continuation-in-part of assignee's application U.S. Ser. No. 09/115,643, filed on Jul. 15, 1998, and entitled "Load **Balancing,**" now U.S. Pat. No. 6,249,801.

Brief Summary Text - BSTX (2):
The present invention relates to computer networks in general, and in particular to load **balancing client** requests among redundant network **servers** in different geographical locations.

Brief Summary Text - BSTX (4):
In computer networks, such as the Internet, preventing a **server** from becoming overloaded with requests from **clients** may be accomplished by providing several **servers** having redundant capabilities and managing the distribution of **client** requests among the **servers** through a process known as "load **balancing.**"

Brief Summary Text - BSTX (5):
In one early implementation of load **balancing,** a Domain Naming System (DNS) server connected to the Internet is configured to maintain several IP addresses for a single domain name, with each address corresponding to one of several servers having redundant capabilities. The DNS **server** receives a request for address translation and responds by returning the list of **server** addresses from which the **client** chooses one address at random to connect to. Alternatively, the DNS server returns a single address chosen either at random or in a round-robin fashion, or actively monitors each of the servers and returns a single address based on server load and availability.

Brief Summary Text - BSTX (6):
More recently, a device known as a "load **balancer,**" such as the Web Server

Director, commercially available from the Applicant/assignee, has been used to **balance** server loads as follows. The load **balancer** is provided as a gateway to several redundant servers typically situated in a single geographical location and referred to as a "server farm" or "server cluster." DNS servers store the IP address of the load **balancer** rather than the addresses of the servers to which the load **balancer** is connected. The load **balancer's** address is referred to as a "virtual IP address" in that it masks the addresses of the servers to which it is connected. **Client requests are addressed** to the virtual IP address of the load **balancer** which then sends the request to a **server** based on **server** load and availability or using other known techniques.

Brief Summary Text - BSTX (7):
Just as redundant **servers** in combination with a load **balancer** may be used to prevent **server** overload, redundant **server** farms may be used to reroute **client** requests received at a first load **balancer/server** farm to a second load **balancer/server** farm where none of the **servers** in the first **server** firm are available to tend to the request. One rerouting method currently being used involves sending an HTTP redirect message from the first load **balancer/server** farm to the **client** instructing the **client** to reroute the request to the second load **balancer/server** farm indicated in the redirect message. This method of load **balancing** is disadvantageous in that it can only be employed in response to HTTP requests, and not for other types of requests such as FTP requests. Another rerouting method involves configuring the first load **balancer** to act as a DNS server. Upon receiving a DNS request, the first load **balancer** simply returns the virtual IP address of the second load **balancer**. This method of load **balancing** is disadvantageous in that it can only be employed in response to DNS requests where there is no guarantee that the request will come to the first load **balancer** since the request does not come directly from the **client,** and where subsequent requests to intermediate DNS **servers** may result in a previously cached response being returned with a virtual IP address of a load **balancer** that is no longer available.

Brief Summary Text - BSTX (8):
Where redundant **server** farms are situated in more than one geographical location, the geographical location of a **client** may be considered when determining the load **balancer to which the client's** requests should be routed, in addition to employing conventional load **balancing** techniques. However, routing **client** requests to the geographically nearest **server,** load **balancer, or server** farm might not necessarily provide the **client** with the best service if, for example, routing the request to a geographically more distant location would otherwise result in reduced latency, fewer hops, or provide more processing capacity at the **server**.

Brief Summary Text - BSTX (10):
The present invention seeks to provide novel apparatus and methods for load **balancing client** requests among redundant network **servers and server** farms in different geographical locations which overcome the known disadvantages of the prior art as discussed above.

Brief Summary Text - BSTX (11):
There is thus provided in accordance with a preferred embodiment of the present invention a method for load **balancing** requests on a network, the method including receiving a request from a requestor having a requestor network address at a first load **balancer** having a first load **balanc r** network address, the request having a source address indicating the requestor network address and a destination address indicating the first load **balancer** network address, forwarding the request from the first load **balancer** to a second load **balancer** at a triangulation network address, the request source address indicating the requestor network address and the destination address indicating the triangulation network address, the triangulation network address being associated with the first load **balancer** network address, and sending a response from the second load **balancer** to the requester at the requestor network address, the response having a source address indicating the first load **balancer** network address associated with the triangulation network address and a destination address indicating the first requestor network address.

Brief Summary Text - BSTX (12):
Further in accordance with a preferred embodiment of the present invention the method includes maintaining the association between the triangulation network address and the first load **balancer** network address at either of the load **balancers**.

Brief Summary Text - BSTX (13):
Still her in accordance with a preferred embodiment of the present invention the method includes maintaining the association between the triangulation network address and the first load **balancer** network address at the second load **balancer,** and communicating the association to the first load **balancer**.

Brief Summary Text - BSTX (14):
Additionally in accordance with a preferred embodiment of the present invention the method includes directing the request from the second load **balancer** to a server in communication with the second load **balancer,** composing the response at the server, and providing the response to the second load **balancer**.

Brief Summary Text - BSTX (15):
There is also provided in accordance with a preferred embodiment of the present invention a method for load **balancing** requests on a network, the method including determining the network proximity of a requestor with respect to each of at least two load **balancers,** designating a closest one of the load **balancers** by ranking the load **balancers** by network proximity, and directing requests from the requestor to the closest load **balancer**.

Brief Summary Text - BSTX (16):
Further in accordance with a preferred embodiment of the present invention the method includes directing requests from any source having a subnet that is

the same as the subnet of the requester to the closest load **balancer**.

Brief Summary Text - BSTX (17):
Still further in accordance with a preferred embodiment of the present invention the method includes monitoring the current load of each of the load **balancers,** and performing the directing step the current load of the closest load **balancer** is less than the curt load o every other of the load **balancers**.

Brief Summary Text - BSTX (22):
Additionally in accordance with a preferred embodiment of the present invention the designating step includes designating a closest one of the load **balancers** by ranking the load **balancers** by network proximity and either of current load and available capacity.

Brief Summary Text - BSTX (23):
There is also provided in accordance with a preferred embodiment of the present invention a method for determining network proximity, the method including sending from each of at least two **servers** a UDP request having a starting TTL value to a **client** at a sufficiently high port number as to elicit an "ICMP port unreachable" reply message to at least one determining one of the **servers** indicating the UDP request's TTL value on arrival at the **client,** determining a number of hops from each of the **servers to the client** by subtract the starting TTL value from the TTL value on arrival for each of the **servers,** and determining which of the **servers** has fewer hops of the **client,** and designating the **server** having fewer hops as being closer to the **client than the other of the servers**.

Brief Summary Text - BSTX (24):
There is additionally provided in accordance with a preferred embodiment of the present invention a network load **balancing** system including a network, a first load **balancer** connected to the network and having a first load **balancer** network address, a second load **balancer** connected to the network and having a triangulation network address, the triangulation network address being associated with the first load **balancer** network address, and a requestor connected to the network and having a requestor network address, where the requestor is operative to send a request via the network to the first load **balancer,** the request having a source address indicating the requestor network address and a destination address indicating the first load **balancer** network address, the first load **balancer** is operative to forward the request to the second load **balancer** at the triangulation network address, the request source address indicating the requestor network address and the destination address indicating the triangulation network address, and the second load **balancer** is operative to send a response to the requestor at the requester network address, the response having a source address indicating the first load **balancer** network address associated with the triangulation network address and a destination address indicating the first requestor network address.

Brief Summary Text - BSTX (25):

Further in accordance with a preferred embodiment of the present invention either of the load **balancers** is operative to maintain a table of the association between the triangulation network address and the first load **balancer** network address.

Brief Summary Text - BSTX (26):
Still further in accordance with a preferred embodiment of the present invention the second load **balancer** is operative to maintain a table of the association between the triangulation network address and the first load bar network address and communicate the association to the first load **balancer**.

Brief Summary Text - BSTX (27):
Additionally in accordance with a preferred embodiment of the present invention the system further includes a server in communication with the second load **balancer**, where the second load **balancer** is operative to direct the request from the second load **balancer** to the server, and the server is operative to compose the response and provide the response to the second load **balancer**.

Brief Summary Text - BSTX (28):
There is also provided in accordance with a preferred embodiment of the present invention a network load **balancing** system including a network, at least two load **balancers** connected to the network, and a requestor connected to the network, where each of the at least two load **balancers** is operative to determine the network proximity of the requestor, and at least one of the load **balancers** is operative to designate a closest one of the load **balancers** by ranking the load **balancers** by network proximity and direct requests from either of the requester and a subnet of the requester to the closest load **balancer**.

Brief Summary Text - BSTX (29):
Further in accordance with a preferred embodiment of the present invention the load **balancers** are operative to poll the requester to yield at least two attributes selected from the group consisting of latency, relative TTL, and number of hops to requester.

Brief Summary Text - BSTX (30):
Still further in accordance with a preferred embodiment of the present invention the load **balancers** are operative to poll the requestor using at least two polling methods selected from the group consisting of: pinging, sending a TCP ACK message to the requestor's source address and port, sending a TCP ACK message to the requestor's source address and port 80, and sending a UDP request to a sufficiently high port number as to elicit an "ICMP port unreachable" reply.

Brief Summary Text - BSTX (31):
Additionally in accordance with a preferred embodiment of the present invention at least one of the load **balancers** is operative to designate the

closest one of the load **balancers** by ranking the load **balancers** by network proximity and either of current load and available capacity.

Brief Summary Text - BSTX (32):
It is noted that throughout the specification and claims the term "network proximity" refers to the quality of the relationship between a **client** and a first **server or server** farm as compared with the relationship between the **client** and a second **server or server** farm when collectively considering multiple measurable factors such as latency, hops, and **server** processing capacity.

Brief Summary Text - BSTX (41):
There is also provided in accordance with yet another preferred embodiment of the present a method for managing a computer network connected to the Internet through a plurality of ISPs, includes the steps of: receiving a request from a **client** within a computer network directed to a remote **server** computer, looking up a table entry within a proximity table indexed by an address related to the remote **server** computer, the tables entries of the proximity table containing ratings for a plurality of ISPs, and selecting one of the plurality of ISPs through which to route the **client** request, based on the ratings within the table entry looked up in the proximity table.

Brief Summary Text - BSTX (47):
The computer network may further be a private network, visible externally through a network address translation. Preferably the method may also include the steps of receiving a response from the remote **server** directed to the source IP address designated for the **client** request, and translating the source IP address designated for the **client address** to the IP address for the **client** within the private network.

Brief Summary Text - BSTX (54):
There is also provided in accordance with another preferred embodiment of the present invention, a network management system for managing a computer network connected to the Internet through a plurality of ISPs, including a network controller receiving a **client** request from within a computer network directed to a remote **server** computer, and selecting one of a plurality of ISPs through which to route the **client** request, and a data manager looking up a table entry within a proximity table indexed by an address related to the remote **server** computer, the tables entries of the proximity table containing ratings for a plurality of ISPs. The network controller may also select one of the plurality of ISP based on the rating within the table entry looked up in the proximity table.

Brief Summary Text - BSTX (58):
Moreover in accordance with a preferred embodiment of the present invention, the network controller routes the client request through the selected ISP. Preferably the computer network is a private network, visible externally through a network address translation, and the network controller receives a

response from the remote **server** directed to the source IP address designated for the **client** request, the system further comprising a network address translator translating the source IP address designated for the **client address** to the IP address for the **client** within the private network.

Brief Summary Text - BSTX (99):
Additionally the route selector is operable to configure and use a Decision Parameter Table comprising parameters of the routes. Furthermore, different Decision Parameters are supplied for each respective content type. The Decision Parameter Table also includes at least one of a group of parameter weights comprising; Data packet content; Hops weighting factor, Packet loss factor and **Response time** factor. It is appreciated that a different Decision Parameters is used for each respective content.

Brief Summary Text - BSTX (100):
A Decision Function F.sub.content is calculated for each path from the first node to the second node, based on said Decision Parameter Table. The Decision Function F.sub.content is defined as: F.sub.content =F(Hops weighting factor*Hops count factor; **Response weighting factor*Response time** factor; Path quality weighting factor*Path quality factor, Packet loss weighting factor*Packet loss factor).

Drawing Description Text - DRTX (3):
FIGS. 1A-1C, taken together, are simplified pictorial flow illustrations of a triangulation load **balancing** system constructed and operative in accordance with a preferred embodiment of the present invention;

Drawing Description Text - DRTX (4):
FIGS. 2A-2F, taken together, are simplified pictorial flow illustrations of a network proximity load **balancing** system constructed and operative in accordance with another preferred embodiment of the present invention;

Drawing Description Text - DRTX (5):
FIGS. 3A-3F, taken together, are simplified pictorial flow illustrations of a preferred embodiment of the present invention for managing and load **balancing** a multi-homed network architecture whereby a client is connected to the Internet through multiple ISPs; and

Detailed Description Text - DETX (2):
Reference is now made to FIGS. 1A-1C which, taken together, are simplified pictorial flow illustrations of a triangulation load **balancing** system constructed and operative in accordance with a preferred embodiment of the present invention. Two server farms, generally designated 10 and 12 respectively, are shown connected to a network 14, such as the Internet, although it is appreciated that more than two server farms may be provided. Server farms 10 and 12 typically comprise a load **balancer** 16 and 18 respectively, which may be a dedicated load **balancer** or a server or router

configured to operate as a load **balanc r,** with each of the load **balancers** being connected to one or more servers 20. Load **balancers** 16 and 18 are alternatively referred to herein as LB1 and LB2 respectively. LB1 and LB2 typically maintain a server status table 22 and 24 respectively, indicating the current load, configuration, availability, and other server information as is common to load **balancers**. LB1 and LB2 also typically periodically receive and maintain each other's overall status and load statistics such that LB1 and LB2 can know each other's availability.

Detailed Description Text - DETX (3):
Typical operation of the triangulation load **balancinq** system of FIGS. 1A-1C is now described by way of example. As is shown more particularly with reference to FIG. 1A, a client 26, such as any known computer terminal configured for communication via network 14, is shown sending a request 28, such as an FTP or HTTP request, to LB1 whose virtual IP address is 100.100.1.0. In accordance with network transmission protocols, request 28 indicates the source IP address of the requestor, being the IP address 197.1.33.5 of client 26, and the destination IP address, being the virtual IP address 100.100.1.0 of LB1. LB2 preferably periodically sends a status report 30 to LB1, the virtual IP address 100.100.1.0 of LB1 being known in advance to LB2. Status report 30 typically indicates the availability of server farm 12 and provides load statistics, which LB1 maintains.

Detailed Description Text - DETX (4):
LB2 is preferably capable of having multiple virtual IP addresses as is well known. It is a particular feature of the present invention for LB2 to designate a currently unused virtual IP address, such as 200.100.1.1, for LB1's use and store the mapping between the IP address of LB1 and the designated IP address in a triangulation mapping table 32, as is shown more particularly with reference to FIG. 1B. The designated address is referred to herein as the triangulation address and may be preconfigured with LB1 or periodically provided to LB1 from LB2. LB1 preferably maintains in a client mapping table 36 a mapping of the IP address 197.1.33.5 of client 26 and the triangulation **address 200.100.1.1 of LB2 to which client 26's requests may be redirected**.

Detailed Description Text - DETX (5):
As shown in the example of FIG. 1A, **server** status table 22 of LB1 indicates that no **servers in server** farm 10 are available to service **client** 26's request, but indicates that **server** farm 12 is available. Having decided that client 26's request should be forwarded to LB2, in FIG. 1C LB1 substitutes the destination IP address of request 28 with the virtual IP address 200.100.1.1 of LB2 which is now mapped to the IP address of client 26 as per client mapping table 36 and sends an address-modified client request 38 to LB2. LB2, upon receiving request 38 at its virtual IP address 200.100.1.1, checks triangulation mapping table 32 and finds that virtual IP address 200.100.1.1 has been designated for LB1's use. LB2 therefore uses the virtual IP address 100.100.1.0 of LB1 as per triangulation mapping table 32 as the source IP address of an outgoing response 40 that LB2 sends to **client** 26 after the request has been serviced by one of the **servers in server** arm 12 selected by LB2. It is appreciated that response 40 must appear to client 26 to come from

LB1, otherwise client 26 will simply ignore response 40 as an unsolicited packet. Client 26 may continue to send requests to LB1 which LB1 then forwards requests to LB2 at the designated triangulation address. LB2 directs requests to an available **server** and sends responses to **client** 26 indicating LB1 as the source IP address.

Detailed Description Text - DETX (6):
Reference is now made to FIGS. 2A-2F which, taken together, are simplified pictorial flow illustrations of a network proximity load **balancing** system constructed and operative in accordance with another preferred embodiment of the present invention. The configuration of the system of FIGS. 2A-2F is substantially similar to FIGS. 1A-1C except as otherwise described hereinbelow. For illustration purposes, a third server farm, generally designated 50, is shown connected to network 14, although it is appreciated that two or more server farms may be provided. Server farm 50 typically comprises a load **balancer** 52, which may be a dedicated load **balancer** or a server or router configured to operate as a load **balancer,** with load **balancer** 52 being connected to two or more servers 20. Load **balancer** 52 is alternatively referred to herein as LB3.

Detailed Description Text - DETX (7):
Typical operation of the network proximity load **balancing** system of FIGS. 2A-2F is now described by way of example. As is shown more particularly with reference to FIG. 2A, client 26 is shown sending request 28, such as an FTP or HTTP request to LB1 whose virtual IP address is 100.100.1.0. LB1 preferably maintains a proximity table 54 indicating subnets and the best server farm site or sites to which requests from a particular subnet should be routed. Determining the "best" site is described in greater detail hereinbelow.

Detailed Description Text - DETX (8):
Upon receiving a request, LB1 may decide to service the request or not based on normal load **balancing** considerations. In any case, LB1 may check proximity table 54 for an entry indicating the subnet corresponding to the subnet of the source IP address of the incoming request. As is shown more particularly with reference to FIG. 2B, if no corresponding entry is found in proximity table 54, LB1 may send a proximity request 56 to LB2, and LB3, whose virtual IP addresses are known in advance to LB1. Proximity request 56 indicates the IP address of client 26.

Detailed Description Text - DETX (9):
A "network proximity" may be determined for a requester such as **client** 26 with respect to each load **balancer/server** farm by measuring and collectively considering various attributes of the relationship such as latency, hops between **client** 26 and each **server** farm, and the processing capacity and quality of each **server** farm site. To determine comparative network proximity, LB1, LB2, and LB3 preferably each send a polling request 58 to client 26 using known polling mechanisms. While known polling mechanisms included pinging client 26, sending a TCP ACK message to client 26 may be used where pinging would otherwise fail due to an intervening firewall or NAT device filtering out a

polling message. A TCP ACK may be sent to the client's source IP address and port. If the client's request was via a UDP connection, a TCP ACK to the client's source IP address and port 80 may be used. One or both TCP ACK messages should bypass any intervening NAT or firewall and cause client 26 to send a TCP RST message, which may be used to determine both latency and TTL. While TTL does not necessarily indicate the number of hops from the client to the load **balancer,** comparing TTL values from LB1, LB2, and LB3 should indicate whether it took relatively more or less hops.

Detailed Description Text - DETX (11):

Client 26 is shown in FIG. 2D sending a polling response 60 to the various polling requests. The responses may be used to determine the latency of the transmission, as well as the TTL value. LB2 and LB3 then send polling results 62 to LB1, as shown in FIG. 2E. The polling results may then be compared, and LB1, LB2, and LB3 ranked, such as by weighting each attribute and determining a total weighted value for each server farm. Polling results may be considered together with **server** farm capacity and availability, such as may be requested and provided using known load **balancing** reporting techniques or as described hereinabove with reference to FIGS. 1A and 1B, to determine the **server** farm site that is "closest" to **client** 26 and, by extension, the **client's** subnet, which, in the example shown, is determined to be LB2. For example, the closest site may be that which has the lowest total weighted value for all polling, load, and capacity results. LB1 may then store the closest site to the client/subnet in proximity table 54.

Detailed Description Text - DETX (12):

As was described above, a load **balancer** that receives a request from a client may check proximity table 54 for an entry indicating the subnet corresponding to the subnet of the source IP address of the incoming request. Thus, if a corresponding entry is found in proximity table 54, the request is simply routed to the location having the best network proximity. Although the location having the best network proximity to a particular subnet may have already been determined, the load **balancer** may nevertheless decide to forward an incoming request to a location that does not have the best network proximity should a load report received from the best location indicate that the location is too busy to receive requests. In addition, the best network proximity to a particular subnet may be periodically redetermined, such as at fixed times or after a predetermined amount of time has elapsed from the time the last determination was made.

Detailed Description Text - DETX (15):

Reference is now made to FIGS. 3A-3F, which illustrate a preferred embodiment of the present invention for managing and load **balancing** a multi-homed network architecture whereby a client is connected to the Internet through multiple ISPs. As illustrated in FIG. 3A, a client 105 is connected to the Internet 110 through three ISPs, 115, 120 and 125, each having a respective router 130, 135 and 140 to controls the flow of data packets. The system includes a content router 145, operative in accordance with a preferred embodiment of the present invention, to provide efficient connectivity between **client** 105 and Internet **servers, such as server** 150. As illustrated in FIG.

3A, **client** 105 has an IP address of 10.1.1.1 on a private network, and seeks to connect to **server** 150 having an IP address of 192.115.90.1.

Detailed Description Text - DETX (16):
As illustrated in FIG. 3B, ISPs 115, 120 and 125 assign respective IP address ranges to the client network, indicated in FIG. 3B by ranges 20.x.x.x, 30.x.x.x and 40.x.x.x. The first time that **client** 105 connects to **server** 150, content router 145 preferably sends polling requests through each of routers 130, 135 and 140 in order to determine the proximity of **server** 150 to **client** 105. When sending the polling requests, content router 145 assigns respective network addresses 20.1.1.1, 30.1.1.1 and 40.1.1.1 to client 105. Thus three polling requests are sent: one from each of the sources 20.1.1.1, 30.1.1.1 and 40.1.1.1 to destination 192.115.90.1.

Detailed Description Text - DETX (18):
Based on these polling results, content router 145 chooses, for example, router 135 as its first choice for connecting **client** 105 with **server** 150. As illustrated in FIG. 3D, proximity results are stored in a proximity table 155. Specifically, proximity table 155 indicates that router 135 is the first choice for connecting content router 145 to any computer residing on subset 192.115.90. Thus, when a new **client** 160 with IP address 10.2.2.2 on the private network attempts to connect to a **server** 165 with IP address 192.115.90.2, through a content router 145, content router 145 determines from proximity table 155 that the best router to use is router 135.

Detailed Description Text - DETX (20):
As illustrated in FIG. 3F, this ensures that subsequent responses sent back from server 165 will be addressed to IP address 30.1.1.1 and, accordingly, will be routed through ISP 120. Content router 145 in turn uses network address translation (NAT) data to determine that IP address 30.1.1.1 corresponds to private IP address 10.2.2.2, and transmits the responses from **server** 165 back to **client** 160.

Detailed Description Text - DETX (22):
FIG. 4B indicates a NAT mapping table 180, showing that the private IP address 10.3.3.3 for server 170 is translated to IP addresses 20.3.3.3, 30.3.3.3 and 40.3.3.3, respectively, by routers 130, 135 and 140. Content router 145 looks up the subnet entry 192.115.90 in proximity table 155, and identifies router 135 as the first choice for best proximity between **server** 170 and **client** 175. In resolving the DNS request, content router 145 accordingly provides 30.3.3.3 as the IP address for the domain name server, even though the original request indicated a destination IP address of 20.1.1.1. This ensures that requests from **client** 175 are sent to **server** 170 with a destination IP address of 30.3.3.3, which in turn ensures that the **client** requests are transmitted through ISP 120.

Detailed Description Text - DETX (23):
It can be seen from FIGS. 3A-3F that the present invention efficiently

balances the load among the three ISPs 115, 120 and 125 for outgoing connections. Similarly, it can be seen from FIGS. 4A and 4B that the present invention efficiently **balances** the load among the three ISPs 115, 120 and 125 for incoming connections. In the event that the router indicated as first choice for the best proximity connection is unavailable or overloaded, the present invention preferably uses a second choice router instead. Thus the present invention ensures that if an ISP service is unavailable, connectivity to the Internet is nevertheless maintained.

Detailed Description Text - DETX (24):
Referring back to FIG. 3F, suppose for example that ISP 120 is unavailable, and that content router 145 routes the outgoing client request through ISP 125 instead of through ISP 120. In accordance with a preferred embodiment of the present invention, content router 145 routes the outgoing request through ISP 125 and labels the outgoing request with a source IP address of 40.1.1.1. Had content router 145 used ISP 125 but indicated a source IP address of 30.1.1.1, the response from **server** 150 would be directed back through ISP 125, and not be able to get through to **client** 160.

Detailed Description Text - DETX (29):
It is appreciated that the managing of the routing, by the content router 508, typically depends on the following factors: the content type, the number of hops to the destination, the **response time** of the destination, the availability of the path, the costing of the link and the average packet loss in the link.

Detailed Description Text - DETX (30):
In order for the content router 508 to determine the "best" path, a "Decision Parameter Table" is built for each content type. It is appreciated that the content type may vary between the application type and actual content (URL requested, or any other attribute in the packet). The Decision Parameter Table is preferably dependent on the parameters: Data packet content; Hops weighting factor; Packet loss factor and **Response time** factor. Typical values of these parameters are also given in Table 1.

Detailed Description Text - DETX (31):
In addition to the parameters listed in Table 1, the following additional parameters may also be taken into consideration Hops count factor; **Response time** factor; Path quality factor; and Packet loss factor.

Detailed Description Text - DETX (32):
A Destination Table is built to summarize the following factors: the content type, the number of hops to the destination, the **response time** of the destination, the availability of the path, and the average packet loss in the link, based on proximity calculations, as previously defined.

Detailed Description Text - DETX (33):

Using the relevant data, as typically listed in Table 1, the content router 508 determines a Decision Function F.sub.content for each path: F.sub.content =F(Hops weighting actor*Hops count factor, **Resp nse weighting factor*Response tim** factor, Path quality weighting factor*Path quality factor, Packet loss weighting factor*Packet loss factor).

Detailed Description Text - DETX (35):
Based on the Decision Function the content router 508 selects one of the available pats. The data packet is then routed through the selected path. The Decision Function for a particular path is determined by an administrative manager (not shown) and may depend, for example, on the minimum number of hops or on the relevant **response time,** or on the packet loss, or on the path quality, or any combination of the above parameters, according to the administrative preferences.

Detailed Description Text - DETX (38):
Thus it may be appreciated that the present invention enables a multi-homed network architecture to realize the full benefits of its redundant route connections by maintaining fault tolerance and by **balancing** the load among these connections, and preferably using data packet content information in an intelligent decision making process.

Claims Text - CLTX (1):
1. A method for managing a computer network connected to the Internet through a plurality of routes, comprising the steps of: receiving a request from a **client** within a **client** computer network directed to a remote **server** computer within a second computer network; looking up a table entry within a proximity table indexed by an address related to the remote **server** computer, the tables entries of the proximity table containing ratings for a plurality of routes between the **client** computer network and the second computer network; and selecting one of the plurality of routes through which to route the client request, based on the ratings within the table entry looked up in the proximity tables, wherein the plurality of routes assign respective IP addresses to the computer network, and wherein the method further comprises the step of setting the source IP address of the client request corresponding to the selected route on the client side.

Claims Text - CLTX (7):
7. The method of claim 6 further comprising the steps of: receiving a response from the remote **server** directed to the source IP address designated for the **client** request; and translating the source IP address designated for the client request to the IP address for the client within the private network.

Claims Text - CLTX (8):
8. A network management system for managing a computer network connected to the Internet through a plurality of routes, comprising: a network controller receiving a **client** request from within a **client** computer network directed to a remote **server** computers, within a second computer network and selecting one of

a plurality of routes through which to route the **client** request; and a data manager looking up a table entry within a proximity table indexed by an address related to the remote **server** computer, the tables entries of the proximity table containing ratings for a plurality of routes, between the **client** computer network and the second computer network and wherein said network controller selects one of the plurality of routes based on the ratings within the table entry looked up in the proximity tables, wherein the plurality of routes assign respective IP addresses to the computer network, and wherein said network controller sets the source IP address of the **client** request corresponding to the selected route on the **client** side.

Claims Text - CLTX (14):

14. The network management system of claim 13 wherein said network controller receives a response from the remote **server** directed to the source IP address designated for the **client** request, the system further comprising a network address translator translating the source IP address designated for the **client** request to the IP address for the **client** within the private network.

Other Reference Publication - OREF (3):

Brochure: "WSD-NP A Powerful Global Load **Balancing** Solution", RadWare Ltd., 1997.

Other Reference Publication - OREF (8):

J. Taschek, "A Well-**Balanced** Web", Internet Computing, 1998.

DOCUMENT-IDENTIFIER: US 20010003823 A1

TITLE: METHOD FOR DOWNLOADING A WEB PAGE TO A CLIENT FOR EFFICIENT DISPLAY ON A TELEVISION SCREEN

---------- KWIC ---------

Abstract Paragraph - ABTX (1):

An improved method of providing a document to a client coupled to a **server**. The **server** provides a number of Internet services to the client, including functioning as a caching proxy on behalf of the client for purposes of accessing the World Wide Web. The proxying **server** retrieves from a remote **server** in response to a request from the client a Web document used to generate a Web page on a television screen coupled to the client. Prior to downloading the requested Web page to the **client, the server** lays out the entire Web page and separates the Web page into partitions such that each one of the partitions corresponds to the viewable display area of the television screen coupled to the client. The server downloads the HTML data that drives the layout within the viewable display area of the television screen. The **server** then downloads all of the image data that is displayed within the viewable display area of the television screen such that the portion of the Web page within the viewable area of the television can be fully generated and displayed by the client in a reduced amount of time. The **server** subsequently downloads the remaining partitions of the Web page in similar fashion until the entire web page has been downloaded to the client.

Cross Reference to Related Applications Paragraph - CRTX (2):

[0002] U.S. patent application entitled, "Method and Apparatus for Managing Communications Between a **Client and a Server** in a Network," having U.S. patent application Ser. No. 08/660,087, and filed on Jun. 3, 1996;

Summary of Invention Paragraph - BSTX (8):

[0012] An improved method for providing a document to a client coupled to a **server** is disclosed. In the method, a document is provided to a proxying server. The document includes image data and non-image data that causes the client to generate a display. The document is partitioned into a plurality of partitions. The data of the first partition is downloaded to the client after the data of the first partition is downloaded, the data of the next partition is downloaded to the client. These steps are repeated until each one of the plurality of partitions has been downloaded to the client. Other features of the present invention will be apparent from the accompanying drawings and from the detailed description which follows method is described of providing a document to a client coupled to a **server**.

Brief Description of Drawings Paragraph - DRTX (3):

[0014] FIG. 1 illustrates several clients connected to a proxying **server** in

a network.

Detail Description Paragraph - DETX (20):
[0042] Referring now to FIG. 5, the **server** 5 initially receives a document request from a client 1 (step 501). The document request will generally result from the user of the client 1 activating a hypertext anchor (link) on a Web page. The act of activating a hypertext anchor may consist of clicking on underlined text in a displayed Web page using a mouse, for example. The document request will typically (but not always) include the URL (Uniform Resource Locator) or other address of the selected anchor. Upon receiving the document request, the server 5 optionally accesses the document database 62 to retrieve stored information relating to the requested document (step 502). It should be noted that the document database 62 is not necessarily accessed in every case. The information retrieved from the document database 62 is used by the server 5 for determining, among other things, how long a requested document has been cached and/or whether the document is still valid. The criteria for determining validity of the stored document are discussed below. The server 5 retrieves the document from the cache 65 if the stored document is valid; otherwise, the server 5 retrieves the document from the appropriate remote server 4 (step 503). The server 5 automatically transcodes the document as necessary based on the information stored in the document database 61 (step 503). The transcoding functions are discussed further below.

Detail Description Paragraph - DETX (33):
[0055] (12) a **redirect address** (if appropriate);

Detail Description Paragraph - DETX (53):
[0074] As mentioned above, the WebTV.TM. services provide a transcoder 66, which is used to rewrite certain portions of the code in an HTML document for various purposes. These purposes include: (1) correcting bugs in documents; (2) correcting undesirable effects which occur when a document is displayed by the client 1; (3) improving the efficiency of transmission of documents from the **server** 5 to the client 1; (4) matching hardware decompression technology within the client 1; (5) resizing images to fit on the television set 12; (6) converting documents into other formats to provide compatibility; (7) reducing latency experienced by a client 1 when displaying a Web page with in-line images (images displayed in text); and, (8) altering documents to fit into smaller memory spaces.

Detail Description Paragraph - DETX (54):
[0075] There are three transcoding modes used by the transcoder 66: (1) streaming, (2) buffered, and (3) deferred. Streaming transcoding refers to the transcoding of documents on a line-by-line basis as they are retrieved from a remote **server** 4 and downloaded to the client 1 (i.e., transcoding "on the fly"). Some documents, however, must first be buffered in the WebTV.TM. **server** 5 before transcoding and downloading them to the client 1. A document may need to be buffered before transmitting it to the client 1 if the type of changes to be made can only be made after the entire document has been retrieved from the remote **server** 4. Because the process of retrieving and

downloading a document to the client 1 increases latency and decreases throughput, it is not desirable to buffer all documents. Therefore, the transcoder 66 accesses and uses information in the document database 61 relating to the requested document to first determine whether a requested document must be buffered for purposes of transcoding, before the document is retrieved from the remote server 4.

Detail Description Paragraph - DETX (58):
[0078] There are countless possibilities of bugs or quirks which might be encountered in a Web document. Therefore, no attempt will be made herein to provide an exhaustive list. Nonetheless, some examples may be useful at this point. Consider, for example, an HTML document that is downloaded from a remote **server** 4 and which contains a table having a width specified in the document as "0." This condition might cause a failure if the client were to attempt to display the document as written. This situation therefore, can be detected and corrected by the transcoder 66. Another example is a quirk in the document which causes quotations to be terminated with too many quotation marks. Once the quirk is first detected and an algorithm is written to recognize it, the transcoder 66 can automatically correct the quirk in any document.

Detail Description Paragraph - DETX (60):
[0080] FIG. 6 illustrates a routine for transcoding a Web document for purposes of eliminating bugs and quirks. Initially, the **server** 5 receives a document request from the client 1 (step 601). Next, the document database 61 is accessed to determine whether or not the requested document has been previously retrieved (step 602). If the document has not been previously retrieved, then the server 5 retrieves the document from the remote server 4 (step 609). Next, the retrieved document is analyzed for the presence of bugs or unusual conditions (step 610). Various diagnostic information is then stored in the document database 61 as a result of the analysis to note any bugs or quirks that were found (step 611). If any bugs or quirks were found which can be corrected by the transcoder 66, the document is then transcoded and saved to the proxy cache 65 (step 612). The transcoded document is then downloaded to the client 1 (step 613). It should be noted that transcoding can be deferred until after the document has been downloaded, as described above; hence, the sequence of FIG. 6 is illustrative only.

Detail Description Paragraph - DETX (61):
[0081] If (in step 602) the requested document had been previously retrieved, then it is determined whether the requested document is still valid (step 603) and whether the document is present in the proxy cache 65 (step 604). If the document is no longer valid, then the document is retrieved from the remote **server** 4, analyzed for bugs and quirks, transcoded as required, and then downloaded to the client 1 as described above (steps 610-613, step 607). Methods for determining validity of a document are discussed below. If the document is still valid (step 603) and the document is present in the cache 65, the document is downloaded to the client 1 in its current form (as it is stored in the cache), since it has already been transcoded (step 608).

Detail Description Paragraph - DETX (67):
[0086] Another purpose for transcoding is to allow documents requested by a client 1 to be displayed by the client 1 more rapidly. Many HTML documents contain references to "in-line" images, or images that will be displayed in text in a Web page. The normal process used in the prior art to display a Web page having in-line images is that the HTML document referencing the image is first downloaded to the client, followed by the client's requesting the referenced image. The referenced image is then retrieved from the remote **server** on which it is located and downloaded to the client. One problem associated with the prior art, however, is that the speed with which a complete Web page can be displayed to the user is often limited by the time it takes to retrieve in-line images. One reason for this is that it simply takes time to retrieve the image itself after the referencing document has been retrieved. Another reason is that, in the prior art, if the referencing document does not specify the size of the image, the Web page generally cannot be displayed until the image itself has been retrieved. The present invention overcomes these limitations.

Detail Description Paragraph - DETX (69):
[0088] Refer now to FIG. 7A, which illustrates a routine for reducing latency when downloading a document referencing an image to a client 1. Assume that a client 1 sends a request to the **server** 5 for an HTML document containing a reference to an in-line image. Assume further that the size of the image is not specified in the document itself. Initially, the server 5 determines whether that document has been previously retrieved (step 701). If not, the standard initial retrieval and transcoding procedure is followed (step 706), as described in connection with FIG. 6. If, however, the document has been previously retrieved, then the transcoder 66 accesses the size information stored in the document database 61 for the in-line image (step 702). Based on this size information, the HTML document is transcoded such that, when the Web page is initially displayed by the client 1, the area in which the image belongs is replaced by a blank region enveloping the shape of the image. Thus, any in-line image referenced by a document is displayed initially as a blank region. Consequently, the client 1 can immediately display the Web page corresponding to the HTML document even before the referenced image has been retrieved or downloaded (i.e., even before the size of the image is known to the client 1).

Detail Description Paragraph - DETX (70):
[0089] As the transcoded HTML document is downloaded to the client, the image is retrieved from the appropriate remote **server** 4 (step 704). Once the image is retrieved from the remote **server** 4 and downloaded to the client 1, the client 1 replaces the blank area in the Web page with the actual image (step 705).

Detail Description Paragraph - DETX (74):
[0092] As mentioned previously, prior art **servers** generally download all of the HTML data, or non-image data, of a Web page to a client and then subsequently download the image data to the client when displaying a Web page,

without consideration of the viewable display area of the screen. A trade-off with this prior art approach is that it optimizes total Web page throughput at the expense of the time required to update the viewable display area of the screen. Although this problem may not be as noticeable on computers with large monitors having large display areas, this problem is more noticeable on displays with relatively smaller viewer areas, such as for example a television set 12 coupled to WebTV.TM. client 1.

Detail Description Paragraph - DETX (76):
[0094] FIG. 7B is a flow diagram illustrating the steps performed to efficiently download Web page data from server 5 to a WebTV.TM. client 1 for efficient display within the viewable display area of a television screen in accordance with the teachings of the present invention. Assume that WebTV.TM. client 1 sends a request to **server** 5 for a document that is used to generate a Web page. Assume further that the document contains non-image data, such as for example HTML data, and image data. Assume also that the overall size of the Web page is such that the entire Web page cannot be displayed on a single screen of television set 12.

Detail Description Paragraph - DETX (79):
[0097] After the non-image data and image data have been downloaded as described from server 5 to WebTV.TM. client 1 for a first television screen of Web page information, **server** 5 then sequentially repeats the steps of downloading the non-image data and the image data for each of the remaining viewable television screens of the Web page information until all of the television screens of the Web page have been downloaded from **server** 5 to WebTV.TM. client 1. In one embodiment, each remaining television screen of the Web page information is not downloaded until all of the non-image data and image data of a previous television screen Web page have been fully transmitted or downloaded.

Detail Description Paragraph - DETX (86):
[0103] Other uses for transcoding include transcoding audio files. For example, audio may be transcoded into different formats in order to achieve a desired **balance** between memory usage, sound quality, and data transfer rate. In addition, audio may be transcoded from a file format (e.g., an ".AU" file) to a streaming format (e.g., MPEG 1 audio). Yet another use of audio transcoding is the transcoding of MIDI (Musical Instrument Digital Interface) data to streaming variants of MIDI.

Detail Description Paragraph - DETX (87):
[0104] Additionally, documents or images requiring a large amount of memory (e.g., long lists) can be transcoded in order to consume less memory space in the client 1. This may involve, for example, separating a large document or image into multiple sections. For example, the **server** 5 can insert tags at appropriate locations in the original document so that the document appears to the client 1 as multiple Web pages. Hence, while viewing a given page representing a portion of the original document, the user can view the next page (i.e., the next portion of the original document) by activating a button

on the screen as if it were an ordinary hypertext anchor.

Detail Description Paragraph - DETX (89):
[0105] As noted above, the **server** 5 functions as a proxy on behalf of the client 1 for purposes of accessing the Web. The document database 61 is used in various ways to facilitate this proxy role, as will now be described.

Detail Description Paragraph - DETX (95):
[0109] The document database 61 is also used by the server 5 to store prefetching information relating to documents and images. In particular, the database stores, for each document that has been retrieved, a list of images referenced by the document, if any, and their locations. Consequently, the next time a document is requested by a client 1, the images can be immediately retrieved by the **server** 5 (from the cache 65, if available, or from the remote **server** 4), even before the client 1 requests them. This procedure improves the speed with which requested Web pages are downloaded to the client.

Detail Description Paragraph - DETX (96):
[0110] The document database 61 is also used to facilitate a process referred to as "**server**-advised client prefetching." **Server**-advised client prefetching allows the **server** 5 to inform the client 1 of documents or images which are popular to allow the client 1 to perform the prefetching. In particular, for any given document, a list is maintained in the server 5 of the most popular hypertext anchors in that document (i.e., those which have previously received a large number of hits). When that document is requested by the client 1, the **server** 5 provides the client 1 with an indication of these popular links.

Detail Description Paragraph - DETX (98):
[0111] Web pages are sometimes forwarded from the remote server on which they are initially placed to a different location. Under the HTTP (Hypertext Transport Protocol), such forwarding is sometimes referred to as a "redirect." When an HTML document is initially stored on one remote server and then later transferred to another remote server, the first remote server will provide, in response to a request for that document, an indication that the document has been transferred to a new remote server. This indication generally includes a forwarding **address ("redirect address**"), which is generally a URL.

Detail Description Paragraph - DETX (99):
[0112] In the prior art, when a computer requesting a Web page receives a **redirect, it must then submit a new request to the redirect address**. Having to submit a second request and wait for a second **response consumes time** and increases overall latency. Consequently, the present invention uses the document database 61 to store any **redirect address** for each document or image. Any time a redirected document is requested, the server 5 automatically accesses the **redirect address** to retrieve the document. The document or image is provided to the client 1 based on only a single request from the client 1.
The change in location of the redirected document or image remains completely

transparent to the client 1.

Detail Description Paragraph - DETX (100):

[0113] FIG. 9 illustrates a routine performed by the server 5 in accessing documents which may have been forwarded to a new remote server. Initially, the server 5 receives a request for a document, which generally includes an address (step 901). The server 5 then accesses the document database 65 to determine whether there is a **redirect address** for the requested document (step 902). If there is no **redirect address, then the server** 5 accesses a remote **server** 4 based on the address provided in the document request from the client 1 (step 903). Assuming that the remote **server** 4 does not respond to the **server** 5 with a redirect (step 904), the document is retrieved and downloaded to the client 1 by the **server** 5 (step 907). If, however, a **redirect address** was stored in the document database 65 (step 902), then the server 5 accesses the requested document according to the **redirect address** (step 906). Or, if the remote server 4 responded with a **redirect (step 904), then the server 5 saves the redirect address** to the document database 61 (step 905) and accesses the requested document according to the **redirect address** (step 906).

Detail Description Paragraph - DETX (102):

[0114] The document database 65 also stores information relating to the performance of each remote server 4 from which a document is retrieved. This information includes the latency and throughput of the remote server 4. Such information can be valuable in instances where a remote server 4 has a history of responding slowly. For example, when the document is requested, this knowledge can be used by the **server** 5 to provide a predefined signal to the client 1. The client 1 can, in response to the signal, indicate to the user that a delay is likely and give the user the option of canceling the request.

Detail Description Paragraph - DETX (104):

[0115] Although the server 5 generally operates in the proxy mode, it can also enter a "backoff mode" in which the server 5 does not act as a proxy, or the server 5 performs only certain aspects of the normal proxying functions. For example, if the proxy cache 65 is overloaded, then the server 5 can enter a backoff mode in which documents are not cached but are transcoded as required. Alternatively, during times when the **server** 5 is severely overloaded with network traffic, the **server** 5 may instruct the client 1 to bypass the **server** 5 and contact remote **servers** 4 directly for a specified time or until further notice. Or, the **server** 5 can enter a flexible backoff mode in which the client 1 will be instructed to contact a remote **server** 4 directly only for certain Web sites for a limited period of time.

Detail Description Paragraph - DETX (106):

[0116] The WebTV.TM. **server** 5 provides various services to the client 1, such as proxying and electronic mail ("e-mail"). In the prior art, certain difficulties are associated with allowing a client computer access to different services of an Internet service, as will now be explained with reference to FIG. 10.

Detail Description Paragraph - DETX (107):

[0117] FIG. 10 illustrates a client-server system according to one prior art embodiment. The **server** 76 provides various services A, B, and C. The **server** 76 includes a database 71 for storing information on the user's access privileges to services A, B, and C. The client 75 of the embodiment of FIG. 10 accesses any of services A, B, and C by contacting that service directly. The contacted service then accesses the database 71, which stores the access privileges of the client 75, to determine whether the client 75 should be allowed to access that service. Hence, each service provided by the server 76 requires direct access to the database 71. This architecture results in a large number of accesses being made to the database 71, which is undesirable. In addition, the fact that each service independently has access to the database 71 raises security concerns. Specifically, it can be difficult to isolate sensitive user information. The present invention overcomes such difficulties using a technique which is now described.

Detail Description Paragraph - DETX (112):

[0121] Referring again to FIG. 11, in response to a log-on request by a client 1, the log-in service 78 consults the user database 62 to determine if access to the **server** 5 by this particular client 1 is authorized. Assuming access is authorized, the log-in service 78 retrieves certain user information pertaining to this particular client 1 from the user database 62. The log-in service then generates a "ticket" 82, which is an information packet including the retrieved information. The ticket 82 is then provided to the client 1 which requested access.

Detail Description Paragraph - DETX (115):

2. Redundancy of Services and Load **Balancing**

Detail Description Paragraph - DETX (116):

[0124] The present invention also includes certain redundancies in the various services provided by the server 5. In particular, a given service (e.g., e-mail) can be provided by more than one physical or logical device. Each such device is considered a "provider" of that service. If a given provider is overloaded, or if the client 1 is unable to contact that provider, the client 1 can contact any of the other providers of that service. When the **server** 5 receives a log-in request from a client 1, in addition to generating the above-described ticket 82, the log-in service 78 dynamically generates a list of available WebTV.TM. services and provides this list to the client 1.

Detail Description Paragraph - DETX (117):

[0125] The **server** 5 can update the list of services used by any client 1 to reflect services becoming unavailable or services coming on-line. Also, the list of services provided to each client 1 can be updated by the **server** 5 based upon changes in the loading of the **server** 5, in order to optimize traffic on the **server** 5. In addition, a client's list of services can be updated by services other than the log-in service 78, such that one service can effectively introduce another service to the client 1. For example, the e-mail

service may provide a client 1 with the name, port number and IP of its address book service. Thus, one service can effectively, and securely within the same chain of trust, introduce another service to the client 1.

Detail Description Paragraph - DETX (121):
3. Asynchronous Notification to **Clients by Server**

Detail Description Paragraph - DETX (122):
[0129] Another limitation associated with prior art Internet **servers** is the inability to provide asynchronous notification information to the client in the absence of a request from the client to do so. It would be desirable, for example, for a **server** to notify a client on its own initiative when a particular Web page has changed or that a particular service is inaccessible. The **server** 5 of the present invention provides such capability, and the client 1 is configured to receive and decode such notifications. For example, the client 1 can receive updates of its listing of service providers from the **server** 5 at various points in time, as already described. Similarly, if a particular service provider becomes unavailable, that fact will be automatically communicated to the client 1. As another example, if e-mail addressed to the user has been received by the **server** 5, then the **server** 5 will send a message to the client 1 indicating this fact. The client 1 will then notify the user that e-mail is waiting by a message displayed on the television set 12 or by an LED (light emitting diode) built into the housing of WebTV.TM. box 10.

Claims Text - CLTX (2):
1. In a proxying **server** coupled to a client, a method of providing a document to the client, the method comprising the steps of: providing the document to the proxying **server**, the document including image data and non-image data for causing the client to generate a display; partitioning the document into a plurality of partitions; downloading data of a first partition to the client; and repeating a step of downloading data of a next partition to the client, the step of downloading data of the next partition to the client being performed after a step of downloading data of a previous partition to the client, the repeating step repeated until each one of the plurality of partitions has been downloaded to the client.

Claims Text - CLTX (3):
2. The method described in claim 1 wherein the providing step includes the step of retrieving the document from a remote **server** coupled to the proxying **server** in response to a request from the client.

Claims Text - CLTX (10):
9. In a proxying **server** coupled to a client and to a remote **server**, the proxying **server** operating as a proxy on behalf or the client for accessing the remote **server**, a method of providing a document to the client, the method comprising the steps of: retrieving the document from the remote **server** in response to a request from the client, the document including Hypertext Mark-up

Language (HTML) data and image data for causing the client to generate a Web page on a display; downloading HTML data of a first partition of the document to the client, the first partition having a display height corresponding to a viewable display height of the display; downloading image data of the first partition of the document to the client; and, repeating, until the document has been entirely provided to the client, the steps of: downloading HTML data of a next partition to the client, the next partition having a display height corresponding to the viewable display height of the display; and, downloading image data of the next partition to the client.

US-PAT-NO: 6470389

DOCUMENT-IDENTIFIER: US 6470389 B1

TITLE: Hosting a network service on a cluster of servers using a single-address image

---------- KWIC ---------

Abstract Text - ABTX (1):

Methods and apparatus for hosting a network service on a cluster of servers, each including a primary and a secondary Internet Protocol (IP) address. A common cluster address is assigned as the secondary address to each of the servers in the cluster. The cluster address may be assigned in UNIX-based servers using an ifconfig alias option, and may be a ghost IP address that is not used as a primary address by any server in the cluster. **Client** requests directed to the cluster address are dispatched such that only one of the **servers** of the cluster responds to a given **client** request. The dispatching may use a routing-based technique, in which all **client** requests directed to the cluster address are routed to a dispatcher connected to the local network of the **server** cluster. The dispatcher then applies a hash function to the **client IP address** in order to select one of the **servers** to process the request. The dispatching may alternatively use a broadcast-based technique, in which a router broadcasts **client** requests having the cluster address to all of the **servers** of the cluster over a local network. The **servers** then each provide a filtering routine, which may involve comparing a **server** identifier with a hash value generated from a **client address,** in order to ensure that only one **server** responds to each request broadcast by the router.

Application Filing Date - AD (1):

19970314

Brief Summary Text - BSTX (2):

The present invention relates generally to data communication networks such as the Internet and more particularly to techniques for hosting network services on a cluster of **servers** used to deliver data over a network in response to **client** requests, where the cluster of **servers** can be collectively identified by a **client** using a single-address image.

Brief Summary Text - BSTX (4):

With the explosive growth of the World Wide Web, many popular Internet web sites are heavily loaded with client requests. For example, it has been reported in S. L. Garfinkel, "The Wizard of Netscape," Webserver Magazine, July/August 1996, pp. 59-63, that home pages of Netscape Communications receive more than 80 million client requests or "hits" per day. A single server hosting a service is usually not sufficient to handle this type of

aggressive growth. As a result, clients may experience slow **response times** and may be unable to access certain web sites. Upgrading the servers to more powerful machines may not always be cost-effective. Another common approach involves deploying a set of machines, also known as a cluster, and configuring the machines to work together to host a single service. Such a **server** cluster should preferably publicize only one **server** name for the entire cluster so that any configuration change inside the cluster does not affect **client** applications. The World Wide Web and other portions of the Internet utilize an application-level protocol, known as the Hypertext Transfer Protocol (HTTP), which is based on a **client/server** architecture. The HTTP protocol is described in greater detail in "Hypertext Transfer Protocol--HTTP/1.0," Network Working Group, May 1996, <http://www.ics.uci.edu/pub/ ietf/http>, which is incorporated by reference herein.

Brief Summary Text - BSTX (5):

FIG. 1 illustrates an exemplary **client/server** architecture suitable for implementing HTTP-based network services on the Internet. A **client** 12 generates an HTTP request for a particular service, such as a request for information associated with a particular web site, and a Transmission Control Protocol/Internet Protocol (TCP/IP) connection is then established between the **client** 12 and a **server** 14 hosting the service. The **client** request is delivered to the **server** 14 in this example via a TCP/IP connection over a first network 16, a router 18 and a second network 20. The first network 16 may be a wide area communication network such as the Internet, while the second network 20 may be an Ethernet or other type of local area network (LAN) interconnecting server 14 with other servers in a server cluster. The router 18, also referred to as a gateway, performs a relaying function between the first and second networks which is transparent to the client 12.

Brief Summary Text - BSTX (6):

The **client** request is generated by a web browser or other application-layer program operating in an application layer 22-1 of the **client** 12, and is responded to by a file transfer system or other program in an application layer 22-2 of the **server** 14. The requested network service may be designated by a Uniform Resource Locator (URL) which includes a domain name identifying the server 14 or a corresponding server cluster hosting the service. The application-level program of the **client** 12 initiates the TCP/IP connection by requesting a local or remote Domain Name Service (DNS) to map the **server** domain name to an IP address. The TCP and IP packet routing functions in **client** 12 and **server** 14 are provided in respective TCP layers 24-1, 24-2 and IP layers 26-1, 26-2. The TCP and IP layers are generally associated with the transport and network layers, respectively, of the well-known Open Systems Interconnection (OSI) model. The TCP layers 24-1, 24-2 process TCP packets of the **client** request and **server** response. The TCP packets each include a TCP header identifying a port number of the TCP connection between the **client** 12 and **server** 14. The IP layers 26-1, 26-2 process IP packets formed from the TCP packets of the TCP layers. The IP packets each include an IP header identifying an IP address of the TCP/IP connection between the **client** 12 and **server** 14.

Brief Summary Text - BSTX (7):

The IP address for a given network service may be determined, as noted above, by the client accessing a conventional DNS. The IP layer 26-1 of the client 12 uses the resulting IP address as a destination address in the IP packet headers of client request packets. The IP address together with the TCP port number provide the complete transport address for the HTTP server process. The **client** 12 and **server** 14 also include data link and physical layers 28-1 for performing framing and other operations to configure **client** request or reply packets for transmission over the networks 16 and 20. The router 18 includes data link and physical layers 28-3 for converting **client** request and **server** reply packets to IP format, and an IP layer 26-3 for performing packet routing based on IP addresses. The **server** 14 responds to a given **client** request by supplying the requested information over the established TCP/IP connection in a number of reply packets. The TCP/IP connection is then closed.

Brief Summary Text - BSTX (8):

There are many known techniques for distributing HTTP **client** requests to a cluster of **servers**. FIGS. 2 and 3 illustrate **server**-side single-IP-address image approaches which present a single IP address to the **clients**. An example of this approach is the TCP router approach described in D. M. Dias, W. Kish, R. Mukherjee and R. Tewari, "A Scalable and Highly Available Web Server," Proceedings of COMPCON '96, pp.85-92,1996, which is incorporated by reference herein. FIG. 2 illustrates the TCP router approach in which a **client** 12 establishes a TCP/IP connection over Internet 30 with a **server**-side router 32 having an IP address RA. The router 32 is connected via a LAN 36 to a server cluster 34 including N servers 14-i, i =1, 2, . . . N, having respective IP addresses S1, S2, . . . SN. Each server of the cluster 34 generally provides access to the same set of contents, and the contents may be replicated on a local disk of each server, shared on a network file system, or served by a distributed file system.

Brief Summary Text - BSTX (9):

The single-address image is achieved by publicizing the address RA of the **server**-side router 32 to the **clients** via the DNS. The client 12 therefore uses RA as a destination IP address in its request. The request is directed to the router 32, which then dispatches the request to a selected **server** 14-k of **server** cluster 34 based on load characteristics, as indicated by the dashed line connecting **client** 12 to **server** 14-k via router 32. The router 32 performs this dispatching function by changing the destination IP address of each incoming IP packet of a given **client** request from the router address RA to the address Sk of selected **server** 14-k. The selected **server** 14-k responds to the **client** request by sending reply packets over the established TCP/IP connection, as indicated by the dashed line connecting **server** 14-k to **client** 12. In order to make the TCP/IP connection appear seamless to the **client** 12, the selected **server** 14-k changes the source IP address in its reply packets from its address Sk to the router address RA. The advantages of this approach are that it does not increase the number of TCP connections, and it is totally transparent to the clients. However, since the above-noted source IP address change is performed at the IP layer in a given server, the kernel code of every server in the cluster has to be modified to implement this mechanism. A proposed hybrid of the DNS approach and the TCP router approach, in which a DNS server selects

one of several clusters of servers using a round-robin technique, suffers from the same problem.

Brief Summary Text - BSTX (10):
FIG. 3 illustrates a server-side single-address image approach known as network address translation, as described in greater detail in E. Anderson, D. Patterson and E. Brewer, "The Magicrouter, an Application of Fast Packet Interposing," Symposium on Operating Systems Design and Implementation, OSDI, 1996, <http://www.cs.berkeley.edu/.about.eanders/magicrouter/ osdi96-mr-submission.ps>, and Cisco Local Director, <http://www.cisco.com/warp/public/751/lodir/index. html>, which are incorporated by reference herein. As in the TCP router approach of FIG. 2, the **client** 12 uses the router address RA as a destination IP address in a **client** request, and the router 32 dispatches the request to a selected **server** 14-k by changing the destination IP address of each incoming request packet from the router address RA to the address Sk of selected **server** 14-k. However, in the network address translation approach, the source IP addresses in the reply packets from the selected **server** 14-k are changed not by **server** 14-k as in FIG. 2, but are instead changed by the router 32. The reply packet flow indicated by a dashed line in FIG. 2 thus passes from **server** 14-k to **client** 12 via router 32.

Brief Summary Text - BSTX (12):
It is therefore apparent that a need exists for improved techniques for hosting a network service on a cluster of **servers** while presenting a single-address image to the **clients,** without the problems associated with the above-described conventional approaches.

Brief Summary Text - BSTX (14):
The present invention provides methods and apparatus for hosting a network service on a cluster of servers. All of the servers in a server cluster configured in accordance with the invention may be designated by a single cluster address which is assigned as a secondary address to each server. All **client** requests for a web site or other network service associated with the cluster address are sent to the **server** cluster, and a dispatching mechanism is used to ensure that each **client** request is processed by only one **server** in the cluster. The dispatching may be configured to operate without increasing the number of TCP/IP connections required for each client request. The invention evenly distributes the **client** request load among the various **servers** of the cluster, masks the failure of any **server or servers** of the cluster by distributing **client** requests to the remaining **servers** without bringing down the service, and permits additional **servers** to be added to the cluster without bringing down the service. Although well-suited for use in hosting web site services, the techniques of the present invention may also be used to support a wide variety of other server applications.

Brief Summary Text - BSTX (15):
In an exemplary embodiment of the invention, a network service is hosted by a server cluster in which each server includes a primary IP address and a

secondary IP address. A common cluster address is assigned as the secondary IP address for each of the servers. The cluster address may be an IP address which does not correspond to a primary IP address of any of the servers. In UNIX-based servers, the cluster address may be assigned as the secondary address for a given server using an ifconfig alias option. If a given server includes multiple network interface cards, the cluster address may be assigned to one of the network interface cards using a UNIX ifconfig command without the alias option, or other similar technique. A router is coupled to a local network of the **server** cluster and is also coupled via the Internet to a **client**. The router receives client requests from the Internet, and uses a dispatching technique to direct client requests having the cluster address as a destination. The **client** requests are dispatched such that each of the requests is processed by only one of the **servers** in the cluster. The dispatching function may be based on the result of applying a hash function to an IP address of the given client. A suitable hash function may be determined using an analysis of a distribution of **client IP addresses** in an access log associated with one or more of the **servers**. In the event that a **server** has failed, the hash function may be reapplied to the **client IP address** to identify another **server**.

Brief Summary Text - BSTX (16):
Two illustrative dispatching techniques for providing a single-address image for a server cluster in accordance with the invention include routing-based dispatching and broadcast-based dispatching. In the routing-based technique, a dispatcher is coupled to the router and to a local network of the server cluster. The router directs **client** requests having the cluster address to the dispatcher, and the dispatcher selects a particular one of the **servers** to process a given **client** request based on the result of applying a hash function to the **client address**. In the broadcast-based technique, the router broadcasts **client** requests having the cluster address to each of the **servers** over the local network of the **server** cluster. Each of the **servers** implements a filtering routine to ensure that each **client** request is processed by only one of the **servers**. The filtering routine may involve applying a hash function to the **client IP address** associated with a given **client** request, and comparing the result to a **server** identifier to determine whether that **server** should process the **client** request.

Brief Summary Text - BSTX (17):
The techniques of the present invention provide fast dispatching and can be implemented with reduced cost and complexity. The techniques are suitable for use in TCP/IP networks as well as networks based on a variety of other standards and protocols. Unlike the conventional single-address image approaches, the present invention does not require that a destination address in a request packet header be changed to a **server** address so that the **server** can accept the request, or that a source address in a reply packet header be changed to the router address so that the **client** can accept the reply. In addition, the router need not store an IP address mapping for every IP connection, nor is it required to search through such a mapping to determine which server a packet should be forwarded to. The router itself will therefore not become a bottleneck under heavy load conditions, and special router hardware designs are not required. These and other features and advantages of

the present invention will become more apparent from the accompanying drawings and the following detailed description.

Drawing Description Text - DRTX (2):
FIG. 1 is a block diagram illustrating a conventional **client-serv r** interconnection in accordance with the TCP/IP standard;

Detailed Description Text - DETX (2):
The present invention will be illustrated below in conjunction with exemplary **client/ server** connections established over the Internet to a **server** cluster using the Transmission Control Protocol/Internet Protocol (TCP/IP) standard. It should be understood, however, that the invention is not limited to use with any particular type of network or network communication protocol. The disclosed techniques are suitable for use with a wide variety of other networks and protocols. The term "server cluster" as used herein refers to a group or set of servers interconnected or otherwise configured to host a network service. The terms "cluster address" and "single-address image" refer generally to an address associated with a group of servers configured to support a network service or services. A "ghost IP address" is one type of cluster address in the form of an IP address which is not used as a primary address for any server of a given server cluster. The term "network service" is intended to include web sites, Internet sites and data delivery services, as well as any other data transfer mechanism accessible by a client over a network. The term "client request" refers to a communication from-a client which initiates the network service. A given client request may include multiple packets or only a single packet, depending on the nature of the request.

Detailed Description Text - DETX (3):
The present invention provides an improved single-address image approach to distributing **client** requests to **servers of a server** cluster. In a preferred embodiment, the invention allows all servers of a server cluster to share a single common IP address as a secondary address. The secondary address is also referred to herein as a cluster address, and may be established using an ifconfig alias option available on most UNIX-based systems, or similar techniques available on other systems. The cluster address may be publicized to clients using the above-noted Domain Name Service (DNS) which translates domain names associated with Uniform Resource Locators (URLs) to IP addresses. All **client** requests to be directed to a service hosted by the **server** cluster are sent to the single cluster address, and dispatched to a selected one of the **servers** using routing-based or broadcast-based dispatching techniques to be described in greater detail below. Once a **server** is selected, future request packets associated with the same **client** request may be directed to the same **server**. All other communications within the server cluster may utilize primary IP addresses of the servers.

Detailed Description Text - DETX (4):
The above-noted ifconfig alias option is typically used to allow a single server to serve more than one domain name. For example, the ifconfig alias

option allows a single server to attach multiple IP addresses, and thus multiple domain names, to a single network interface, as described in "Two Servers, One Interface" <http://www.thesphere.com/.about.dlp/TwoServers/>, which is incorporated by reference herein. **Client** requests directed to any of the multiple domain names can then be serviced by the same **server**. The server determines which domain name a given request is associated with by examining the destination address in the request packet. The present invention utilizes the ifconfig alias option to allow two servers to share the same IP address. Normally, two **servers** cannot share the same IP address because such an arrangement would cause any packet destined for the shared address to be accepted and responded to by both **servers,** confusing the **client** and possibly leading to a connection reset. Therefore, before a server is permitted to attach a new IP address to its network interface, a check may be made to ensure that no other server on the same local area network (LAN) is using that IP address. If a duplicate address is found, both servers are informed and warnings are issued. The routing-based or broadcast-based dispatching of the present invention ensures that every packet is processed by only one server of the cluster, such that the above-noted warnings do not create a problem.

Detailed Description Text - DETX (5):
An alternative technique for assigning a secondary address to a given server of a server cluster in accordance with the invention involves configuring the given server to include multiple network interface cards such that a different address can be assigned to each of the network interface cards. For example, in a UNIX-based system, conventional ifconfig commands may be used, without the above-described alias option, to assign a primary IP address to one of the network interface cards and a secondary IP address to another of the network interface cards. The secondary IP address is also assigned as a secondary IP address to the remaining **servers** in the cluster, and used as a cluster address for directing **client** requests to the cluster.

Detailed Description Text - DETX (6):
The exemplary embodiments of the present invention to be described below utilize dispatching techniques in which **servers** are selected based on a hash value of the **client IP address**. The hash value may be generated by applying a hash function to the **client IP address,** or by applying another suitable function to generate a hash value from the **client IP address**. For example, given N **servers** and a packet from a **client having a client address** CA, a dispatching function in accordance with the invention may compute a hash value k as CA mod (N-1) and select **server** k to process the packet. This ensures that all request or reply packets of the same TCP/IP connection are directed to the same server in the server cluster. A suitable hash function may be determined by analyzing a distribution of **client IP addresses** in actual access logs associated with the **servers such that client** requests are approximately evenly distributed to all **servers**. When a **server** in the cluster fails, the subset of **clients** assigned to that **server** will not be able to connect to it. The present invention addresses this potential problem by dynamically modifying the dispatching function upon detection of a server failure. If the hash value of a given **client IP address** maps to the failed **server, the client IP address** is rehashed to map to a non-failed **server,** and the connections of the remaining

clients are not affected by the failure.

Detailed Description Text - DETX (7):
FIG. 4 illustrates a routing-based dispatching technique in accordance with the present invention. Solid lines indicate network connections, while dashed lines show the path of an exemplary client request and the corresponding reply. A **client** 52 sends a **client** request to a **server** cluster 54 including N **servers** 54-i, i=1, 2, . . . N having IP addresses S1, S2, . . . SN and interconnected by an Ethernet or other type of LAN 56. The **client** request is formulated in accordance with the above-described HTTP protocol, and may include a URL with a domain name associated with a web site or other network service hosted by the **server** cluster 54. The **client** accesses a DNS to determine an IP address for the domain name of the service, and then uses the IP address to establish a TCP/IP connection for communicating with one of the **servers** 54-i of the **server** cluster 54. In accordance with the invention, a "ghost" IP address is publicized to the DNS as a cluster address for the server cluster 54. The ghost IP address is selected such that none of the servers 54-i of cluster 54 has that IP address as its primary address. Therefore, any request packets directed to the ghost IP address are associated with client requests for the service of the single-address image cluster 54. The use of the ghost IP address thus distinguishes a network service hosted by the cluster from activities of the servers 54-i which utilize the primary server addresses, and prevents interference with these primary address activities.

Detailed Description Text - DETX (8):
The **client** 52 uses the ghost IP address as a cluster address for directing its request to the **server** cluster 54. The request is directed over Internet 60 to a router 62 having an IP address RA. The router 62 includes a routing table having an entry or record directing any incoming request packets having the ghost IP address to a dispatcher 64 connected to the LAN 56. The dispatcher 64 includes an operating system configured to run in a router mode, using a routing algorithm which performs the dispatching described herein. In alternative embodiments, the functions of the dispatcher 64 could be incorporated into the router 62 in order to provide additional efficiency improvements. Each of the servers 54-i of the cluster 54 utilizes the above-described ifconfig alias option to set the ghost IP address as their secondary address. As noted above, this technique for setting a secondary address for each of the servers 54-i generally does not require any alteration of the kernel code running on the servers. In alternative embodiments, one or more of the servers 54-i may be configured to include multiple network interface cards, as previously noted, such that a different address can be assigned to each of the network interface cards of a given server using a UNIX ifconfig command or other similar technique.

Detailed Description Text - DETX (9):
The router 62 routes any packets having the ghost IP address to the dispatcher 64 in accordance with the above-noted routing table record. The dispatcher 64 then applies a hash function to the **client IP address** in a given request packet to determine which of the **servers** 54-i the given packet should be routed to. In the example illustrated in FIG. 4, the dispatcher 64 applies

a hash function to the IP address of **client** 52 and determines that the corresponding request packet should be routed to **server** 54-2 at IP address S2. The dispatcher 64 then routes the request packet to the server 54-2 over LAN 56, as indicated by the dashed line, using the primary address S2 of server 54-2 to distinguish it from the other servers of cluster 54. After the network interface of server 54-2 accepts the packet, all higher level processing may be based on the ghost IP address because that is the destination address in the packet IP header and possibly in the application-layer packet contents. After processing the request, the **server** 54-2 replies directly to the **client** 52 via router 62 over the established TCP/IP connection, using the ghost IP address, and without passing through the dispatcher 64.

Detailed Description Text - DETX (10):

It should be noted that when a request packet destined for the ghost IP address is received by the network interface of dispatcher 64 and placed back onto the same network interface for delivery to one of the servers 54-i over LAN 56, it may cause an Internet control message protocol (ICMP) host redirect message to be sent to the router 62. This ICMP message is designed to direct the router 62 to update its routing table such that any future packets having the ghost IP address can bypass the dispatcher 64 and go directly to the destination server, as described in greater detail in W. R. Stevens, TCP/IP Illustrated, Vol. 1, Ch. 6, pp. 69-83, which is incorporated by reference herein. However, this effect is undesirable in the routing technique of FIG. 4 because the dispatcher 64 performs the server selection process as previously described. It therefore may be necessary to suppress the ICMP host **redirect message for the ghost IP address** by, for example, removing or altering the corresponding operating system code in the dispatcher. In the above-mentioned alternative embodiments in which the dispatching function is implemented within the router 62, the ICMP redirect message is not generated and therefore need not be suppressed. Another potential problem may arise when a reply packet is sent back to the **client** 52 from the selected **server** 54-2 with the ghost IP address, in that it may cause the router 62 to associate, in its Address Resolution Protocol (ARP) cache, the ghost IP address with the LAN address of the selected **server**. The operation of the ARP cache is described in greater detail in W. R. Stevens, TCP/IP Illustrated, Vol. 1, Chs. 4 and 5, pp. 53-68, which is incorporated by reference herein. The illustrative embodiment of FIG. 4 avoids this problem by automatically routing the request packets to the dispatcher 64, and then dispatching based on the server primary IP address, such that the router ARP cache is not used.

Detailed Description Text - DETX (11):

FIG. 5 illustrates a broadcast-based dispatching technique in accordance with the present invention. Again, solid lines indicate network connections, while dashed lines show the path of an exemplary client request and the corresponding reply. As in the FIG. 4 routing-based embodiment, **client** 52 sends a **client** request to **server** cluster 54 including N **servers** 54-i, i =1, 2, . . . N connected to LAN 56 and having IP addresses S1, S2, . . . SN. The **client** 52 uses the above-described ghost address as a cluster address for directing its request to the **server** cluster 54. The request is directed over Internet 60 to a router 70 having an IP address RA. The router 70 broadcasts any incoming request packets having the ghost IP address to the LAN 56

interconnecting the servers 54-i of the server cluster 54, such that the request packet is received by each of the servers 54-i.

Detailed Description Text - DETX (12):

Each of the **servers** 54-i of the cluster 54 implements a filtering routine in order to ensure that only one of the **servers** 54-i processes a given **client** request. The filtering routine may be added to a device driver of each of the servers 54-i. In an exemplary implementation, each of the servers 54-i is assigned a unique identification (ID) number. The filtering routine of a given **server** 54-i computes a hash value of the **client IP address** and compares it to the ID number of the given **server**. If the hash value and the ID number do not match, the filtering routine of the given server rejects the packet. If the hash value and the ID number do match, the given server accepts and processes the packet as if it had received the packet through a conventional IP routing mechanism. In the illustrative example of FIG. 5, a packet associated with the request from **client** 52 is broadcast by the router 70 to each of the **servers** 54-i of the **server** cluster 54 over the LAN 56 as previously noted. The filtering routine of **server** 54-2 generates a hash value of the **client IP address** which matches the unique ID number associated with **server** 14-2, and **server** 14-2 therefore accepts and processes the packet. The filtering routines of the N-1 other **servers** 54-i each indicate no match between the **client IP address and the corresponding server** ID number, and therefore discard the broadcast packet. The reply packets are sent back to the client 52 via router 70, as indicated by the dashed lines, using the ghost IP address.

Detailed Description Text - DETX (14):

The routing-based and broadcast-based dispatching techniques described in conjunction with FIGS. 4 and 5 above have been implemented on a cluster of Sun SPARC workstations. The NetBSD operating system, as described in NetBSD Project, <http://www. NetBSD.org>, was used to provide any needed kernel code modifications. The dispatching overhead associated with both techniques is minimal because the packet dispatching is based on simple IP address hashing, without the need for storing or searching any address-mapping information. In the routing-based dispatching technique, the additional routing step in the dispatcher 64 typically adds a delay of about 1 to 2 msecs to the TCP round-trip time of each incoming request packet. A study in W. R. Stevens, TCP/IP Illustrated, Volume 3, pp. 185-186, which is incorporated by reference herein, indicates that the median TCP round-trip time is 187 msecs. The additional delay attributable to the routing-based dispatching is therefore negligible. Although the additional routing step for every request packet sent to the ghost IP address may increase the traffic in the LAN of the **server** cluster, the size of a request in many important applications is typically much smaller than that of the corresponding response, which is delivered directly to the **client** without the additional routing. In the broadcast-based dispatching technique, the broadcasting of each incoming request packet on the LAN of the server cluster does not substantially increase network traffic. Although a hash value is computed for each incoming packet having the ghost IP destination address, which increases the CPU load of each server, this additional computation overhead is negligible relative to the corresponding communication delay.

Detailed Description Text - DETX (16):

The routing-based and broadcast-based dispatching of the present invention can also provide load **balancing** and failure handling capabilities. For example, given N **servers** and a packet from **client address** CA, the above-described routing-based dispatching function may compute a hash value k as CA mod (N-1) and select **server** k to process the packet. More sophisticated dispatching functions can also be used, and may involve analyzing the actual service access log to provide more effective load **balancing**. In order to detect failures, each server may be monitored by a watchdog daemon such as the watchd daemon described in greater detail in Y. Huang and C. Kintala, "Software Implemented Fault Tolerance: Technologies and Experience," Proceedings of the 23.sup.rd International Symposium on Fault-Tolerant Computing--FTCS, Toulouse, France, pp. 2-9, June 1993, which is incorporated by reference herein. When a server fails, the corresponding watchd daemon initiates a change of the dispatching function to mask the failure and rebalance the load. A system call interface may be implemented to allow the dispatching function to be changed while the servers remain on-line. In routing-based dispatching, the watchd daemon may notify the dispatcher to change the dispatching function, while in broadcast-based dispatching, all servers may be notified to modify their filtering routines. For example, if a server k fails, the new dispatching function may check to see if the hash value CA mod N equals k. If it does, a new hash value j=CA mod (N-1) is computed. If j is less than k, the packet goes to server j. Otherwise, the packet goes to server j+1. This technique does not affect the **clients** of non-failed **servers,** reassigns the **clients** of the failed **server** evenly to the remaining **servers,** and can be readily extended to handle multiple **server** failures. Additional servers can be added to the cluster without bringing down the service by changing the dispatching function from CA mod N to CA mod (N+1).

Detailed Description Text - DETX (18):

The use of the ifconfig alias option or other similar technique to provide a single-address image for a server cluster provides a number of advantages over the conventional techniques described previously. For example, it avoids the need to change the destination address in a request packet header so that a particular **server** can accept the request, and the need to change the source address in a reply packet header to the cluster address so that the **client** can accept the reply. With the single-address image approach of the present invention, all servers can accept and respond to packets having the cluster address, so that the addresses in the request and reply packet headers do not need to be modified. Since the single-image approach of the present invention does not require alteration of the packet addresses, it is suitable for use with a wide variety of protocols, including those protocols which utilize IP addresses within an application program. In addition, the single-address image approach of the present invention does not require a router to store or to search through a potentially large number of IP address mappings in order to determine which cluster server should receive a request packet. The invention thus effectively removes the possibility that the router may become a bottleneck under heavy load conditions.

Claims Text - CLTX (1):

1. A method of routing **client** requests to a plurality of **servers** configured to support a network service over a communication network, each of the **servers** having a primary address, the method comprising the steps of: assigning a common address as a secondary address for each of the plurality of **servers such that each f the servers** individually has the secondary address; and processing **client** requests directed to the common address such that each of the requests is processed by a particular one of the plurality of **servers** without modification of the common address in the request.

Claims Text - CLTX (6):
6. The method of claim 1 wherein the processing step includes the step of dispatching a request of a given **client** to one of the plurality of **servers** based on application of a hash function to an IP address of the given **client**.

Claims Text - CLTX (7):
7. The method of claim 6 wherein the hash function is determined based on an analysis of a distribution of **client IP addresses** in an access log associated with one or more of the **servers**.

Claims Text - CLTX (8):
8. The method of claim 6 wherein the dispatching step includes reapplying the hash function to the **client IP address** to identify another **server if a server** identified as a result of a previous application of the hash function has failed.

Claims Text - CLTX (9):
9. The method of claim 1 wherein the processing step includes the steps of: routing **client** requests directed to the common address to a dispatcher connected to a local network associated with the plurality of **servers**; and selecting a particular one of the **servers** to process a given **client** request based on application of a hash function to a corresponding **client address** in the dispatcher.

Claims Text - CLTX (10):
10. The method of claim 1 wherein the processing step includes the steps of: broadcasting a given **client** request directed to the common address to each of the plurality of **servers** over a local network associated with the **servers**; and implementing a filtering routine in each of the plurality of **servers** so that the given **client** request is processed by only one of the **servers**.

Claims Text - CLTX (11):
11. The method of claim 10 wherein the implementing step includes the steps of: applying a hash function to a **client IP address** associated with the given client request; and comparing the result of the applying step to an identifier of a particular **server** to determine whether that **server** should process the given **client** request.

Claims Text - CLTX (12):
12. An apparatus for routing **client** requests to a plurality of **servers** configured to support a network service over a communication network, each of the **serv rs** having a primary address, the apparatus comprising: means for assigning a common address as a secondary address for each of the plurality of **servers such that each of the servers** individually has the secondary address; and means for processing **client** requests directed to the common address such that each of the requests is processed by a particular one of the plurality of **servers** without modification of the common address in the request.

Claims Text - CLTX (17):
17. The apparatus of claim 12 wherein the processing means is operative to dispatch a request of a given **client** to one of the plurality of **servers** based on application of a hash function to an IP address of the given **client**.

Claims Text - CLTX (18):
18. The apparatus of claim 17 wherein the hash function is determined based on an analysis of a distribution of **client IP addresses** in an access log associated with one or more of the **servers**.

Claims Text - CLTX (19):
19. The apparatus of claim 17 wherein the processing means is further operative to reapply the hash function to the **client IP address** to identify another **server if a server** identified as a result of a previous application of the hash function has failed.

Claims Text - CLTX (20):
20. The apparatus of claim 12 wherein the processing means further includes a dispatcher connected to a local network associated with the plurality of **servers,** wherein the dispatcher is operative to receive **client** requests directed to the common address, and to select a particular one of the **servers** to process a given **client** request based on application of a hash function to a corresponding **client address**.

Claims Text - CLTX (21):
21. The apparatus of claim 12 wherein the processing means further includes: means for broadcasting a given **client** request directed to the common address to each of the plurality of **servers** over a local network associated with the **servers**; and means for filtering the given **client** request in each of the plurality of **servers** so that the given **client** request is processed by only one of the **servers**.

Claims Text - CLTX (22):
22. The apparatus of claim 21 wherein the filtering means is operative to apply a hash function to a **client IP address** associated with the given **client** request, and to compare the result of the applying step to an identifier of a

particular **server** to determine whether that **server** should process the given **client** request.

Claims Text - CLTX (23):

23. An apparatus for routing **client** requests for a network service over a communication network, the apparatus comprising: a plurality of **servers** configured to support the network service, each of the **servers** having a primary address and a secondary address, wherein a common address is assigned as the secondary address for each of the plurality of **servers such that each of the servers** individually has the secondary address; and a router coupled to the **servers** and operative to route **client** requests directed to the common address such that each of the requests is processed by a particular one of the plurality of **servers** without modification of the common address in the request.

Claims Text - CLTX (28):

28. The apparatus of claim 23 wherein the router is further operative to route **client** requests such that a request of a given **client** is routed to one of the plurality of **servers** based on application of a hash function to an IP address of the given **client**.

Claims Text - CLTX (29):

29. The apparatus of claim 28 wherein the hash function is determined based on an analysis of a distribution of **client IP addresses** in an access log associated with one or more of the **servers**.

Claims Text - CLTX (30):

30. The apparatus of claim 28 wherein the hash function is reapplied to the **client IP address** to identify another **server if a server** identified as a result of a previous application of the hash function has failed.

Claims Text - CLTX (31):

31. The apparatus of claim 23 further including a dispatcher coupled to the router and to a local network associated with the plurality of **servers,** such that the router directs **client** requests having the common address to the dispatcher, and the dispatcher selects a particular one of the **servers** to process a given **client** request based on application of a hash function to a corresponding **client address**.

Claims Text - CLTX (32):

32. The apparatus of claim 23 wherein the router is further operative to broadcast a given **client** request directed to the common address to each of the plurality of **servers** over a local network associated with the **servers, and further wherein each of the servers** implements a filtering routine so that the given **client** request is processed by only one of the **servers**.

Claims Text - CLTX (33):

33. The apparatus of claim 32 wherein the filtering routine involves applying a hash function to a **client IP address** associated with the given **client** request, and comparing the result to an identifier of a particular **server** to determine whether that **serv_r** should process the given **client** request.

Claims Text - CLTX (34):

34. A method of routing **client** requests to a plurality of **servers** configured to support a network service over a communication network, each of the **servers** having a primary address, the method comprising the steps of: assigning a common address as a secondary address for each of the plurality of **servers such that each of the servers** individually has the secondary address; and processing **client** requests directed to the common address such that each of the requests is processed by a particular one of the plurality of **servers** without modification of the common address in the request, wherein each of the **servers** provides access to substantially the same set of contents associated with the network service.

Claims Text - CLTX (35):

35. An apparatus for routing **client** requests to a plurality of **servers** configured to support a network service over a communication network, each of the **servers** having a primary address, the apparatus comprising: means for assigning a common address as a secondary address for each of the plurality of **servers such that each of the servers** individually has the secondary address; and means for processing **client** requests directed to the common address such that each of the requests is processed by a particular one of the plurality of **servers** without modification of the common address in the request, wherein each of the **servers** provides access to substantially the same set of contents associated with the network service.

Claims Text - CLTX (36):

36. An apparatus for routing **client** requests for a network service over a communication network, the apparatus comprising: a plurality of **servers** configured to support the network service, each of the **servers** having a primary address and a secondary address, wherein a common address is assigned as the secondary address for each of the plurality of **servers such that each of the servers** individually has the secondary address, and wherein each of the **servers** provides access to substantially the same set of contents associated with the network service; and a router coupled to the **servers** and operative to route **client** requests directed to the common address such that each of the requests is processed by a particular one of the plurality of **servers** without modification of the common address in the request.

Claims Text - CLTX (37):

37. A method of routing **client** requests to a plurality of **servers** configured to support a network service over a communication network, each of the **servers** having a primary address, the method comprising the steps of: assigning a common address as a secondary address for each of the plurality of

servers such that each f the servers individually has the secondary address; broadcasting a given **client** request directed to the common address to each of the plurality of **servers**; and implementing a filtering routine in each of the plurality of **servers** so that the given **client** request is processed by only one of the **servers** without modification of the common address in the request.

Claims Text - CLTX (38):

38. An apparatus for routing **client** requests to a plurality of **servers** configured to support a network service over a communication network, each of the **servers** having a primary address, the apparatus comprising: means for assigning a common address as a secondary address for each of the plurality of **servers such that each of the servers** individually has the secondary address; means for broadcasting a given **client** request directed to the common address to each of the plurality of **servers**; and means for filtering the given **client** request in each of the plurality of **servers** so that the given **client** request is processed by only one of the **servers** without modification of the common address in the request.

Claims Text - CLTX (39):

39. An apparatus for routing **client** requests for a network service over a communication network, the apparatus comprising: a plurality of **servers** configured to support the network service, each of the **servers** having a primary address and a secondary address, wherein a common address is assigned as the secondary address for each of the plurality of **servers such that each of the servers** individually has the secondary address; and a router coupled to the **servers** and operative to route **client** requests directed to the common address such that each of the requests is processed by a particular one of the plurality of **servers,** wherein the router is further operative to broadcast a given **client** request directed to the common address to each of the plurality of **servers, and further wherein each of the servers** implements a filtering routine so that the given **client** request is processed by only one of the **servers** without modification of the common address in the request.

Other Reference Publication - OREF (5):

T. Brisco, "DNS Support for Load **Balancing,**" Network Working Group, RFC 1794, <http://andrew2.andrew.cmu.edu/rfc/rfc1794.html>.

US-PAT-NO: 6314465

DOCUMENT-IDENTIFIER: US 6314465 B1

TITLE: Method and apparatus for load sharing on a wide area network

---------- KWIC ---------

Abstract Text - ABTX (1):

Client's (106-1-106-N, 107-1-107-M) on local area networks (102, 103) making requests to hot sites, which are connected on a wide area network (100) such as the Internet, are redirected through one of a possible plurality of different redirectors (101, 103) to one of a possible plurality of caching **servers** (S1, S2, S3), which each have responsibility for mapping one or more of the hot sites. Each request is probabilistically directed by one of the redirectors to one of the caching **servers** that map the requested hot site in accordance with weights that are determined for that redirector-hot site pair so as to minimize the average delay that all **client** requests across the network will encounter in making requests to all the cached hot sites. In order to determine the weights with which each redirector will redirect requests to the hot sites to the caching servers, statistics of access rates to each hot site are dynamically determined by each redirector in the network from the traffic flow and reported to a central management station (CMS) (115). Network delay is similarly measured by each redirector and reported to the CMS, and server delay is computed using a queuing model of each server. Using these parameters as inputs, a non-linear programming optimization problem is solved as a network flow problem in order to determine the weights for each redirector that will minimize the average delay. As the access rate statistics, as well as the network delay and server delay, dynamically change, the CMS, using the network flow algorithm, recalculates the weights and forwards them back to each redirector. In other embodiments, the redirector-logical item pair for which the redirector probabilistically directs client requests may be other than a hot site identity. For example, the logical items can be groups of **clients** or groups of documents, and the **servers** to which requests are forwarded can be web **servers** or caching **servers**.

Application Filing Date - AD (1):

19990311

Brief Summary Text - BSTX (2):

This invention relates to wide area networks such as the Internet, and more particularly, to a method and apparatus for minimizing the average delay per unit of time of all **client** requests incurred over all connections established for accessing **server** devices, such as web **servers** and proxy cache **servers**, which are located across the wide area network.

Brief Summary Text - BSTX (4):
In co-pending patent application Ser. No. 08/953577 entitled "Data Distribution Techniques for Load-**Balanced** Fault-Tolerant Web Access", by B. Narendran, S. Rangarajan (co-inventor herein) and S. Yajnik, assigned to the assignee of the present application, and which is incorporated herein by reference, a methodology is described for **balancing** the load on a set of devices connected on a wide area network such as the Internet. Specifically, the methodology for load **balancing** described in that application provides, in a first phase, an algorithm for distributing web documents, or objects, onto different servers such that the total access rates to each server (equal to the total number of connection requests that a server handles per time unit) are **balanced** across all the servers. Further, in a second phase of the methodology, a network flow-based algorithm is used to re-**balance** the access rates to each server in response to system changes without moving objects between the different servers.

Brief Summary Text - BSTX (5):
In further detail, in the first phase of the load **balancing** methodology, logical items, such as the web documents, or objects, are mapped to different physical devices such as web servers, cache servers, ftp servers, etc., based on the a priori access rates that are known for requests from/to these web documents. This mapping, referred to the initial distribution, takes as an input the access rates of each web document, the number of replicas of these web documents that need to be made on the physical devices, such as document servers or caches, and the capacity of each of the physical devices, and produces a mapping between the web documents and the physical devices. It also produces as an output the probabilities (or weights) that will then be used by a redirection server to redirect requests from/to replicated web documents to one of the physical devices to which they are mapped. This initial distribution mapping is performed such that the load is **balanced** among the physical devices, or, i.e., the sum of access rates of requests to the web documents redirected to each physical device is **balanced** across all the devices. Load **balance** is achieved across the physical devices irrespective of the web documents that they handle.

Brief Summary Text - BSTX (6):
In the second phase of the methodology, once the initial distribution of the web documents is performed, any change in the system parameters that affects the load **balance** is handled using a network flow load **balance** algorithm to determine new probabilities (or weights) with which the redirection server will then thereafter redirect requests from/to web documents to one of the physical devices to which they are mapped. Thus, instead of re-mapping web documents to different documents servers or caches to handle a perturbation in load, the load is re-**balanced** by changing the probability with which requests to each replicated web document is redirected to one of the plurality of physical devices to which that physical item is mapped. Examples of parameters that may change in the system include the load on each physical device and the capacity of each of the physical devices, the latter of which can instantly become zero upon the failure of a device.

Brief Summary Text - BSTX (7):
The goal of load **balancing, as described, is to balance** across all physical devices the sum of the access rates of requests to the web documents redirected to each physical device. The latency, or delay, incurred in providing a response from a physical device to a request for a web document made by a client has not been previously considered.

Brief Summary Text - BSTX (9):
In accordance with the present invention, minimization of request latency on a wide area network such as the Internet is the goal rather than pure load **balancing**. The load sharing methodology of the present invention minimizes delay by determining the probabilities, or weights with which web requests are redirected over the wide area network to the web servers so as to minimize the average delay of all connections across all servers per unit of time. Such redirection to the different servers is effected as a function of a logical item, the logical item being the factor that the redirector uses in determining where and with what weights the request is to be directed. In determining a solution to a non-linear programming optimization problem, the network delay associated with accessing a server and the server delay, which itself is a function of the access rate on that server, are taken into account. After an initial distribution of logical items is completed, such as for load **balancing** purposes in accordance with the aforedescribed prior art methodology, or another method, the following are determined: (1) the access rates, which are equal to the number of requests per unit time associated with each redirector-logical item pair; (2) the network delay, which is equal to the sum of the propagation and the transmission delays between the **client and the server**; and (3) the **server** delays incurred in processing a web request. Once these parameters are measured or mathematically computed they are used to determine the solution of a non-linear program optimization problem. This non-linear programming problem is formulated and solved as a minimum cost network flow problem to ultimately determine the probability or distributions, or weights, with which each redirector in the network will then redirect requests to the different servers which can satisfy them.

Drawing Description Text - DRTX (2):
FIG. 1 is a block diagram of a system on a wide area network showing two redirectors which direct requests from groups of **clients** for hot sites to plural caching **servers** which are responsible for such hot sites in accordance with a mapped relationship;

Drawing Description Text - DRTX (3):
FIG. 2 is a prior art network flow solution for load **balancing** for a system containing a single redirector with plural caching servers;

Drawing Description Text - DRTX (4):
FIG. 3 is a network flow solution for load **balancing** for a system containing two redirectors;

Detailed Description Text - DETX (2):

The present invention is illustrated below in conjunction with exemplary **client/server** connection s established over the Internet using the Transmission Control Protocol/Internet Protocol (TCP/IP) standard. It should be understood, however, that the invention is not limited to use with any particular type of network or network communication protocol. The disclosed techniques are suitable for use with a wide variety of other networks and protocols. The term "web" as used herein is intended to include the World Wide Web, other portions of the Internet, or other types of communication networks. The term "**client** request" refers to any communication from a **client** which includes a request for information from a **server**. A given request may include multiple packets or only a single packet, depending on the nature of the request. The term "document" as used herein is intended to include web pages, portions of web pages, computer files, or other type of data including audio, video and image data.

Detailed Description Text - DETX (3):

FIG. 1 shows an exemplary web server system 100 in accordance with an illustrative embodiment of the invention. The system includes a first redirection server R1 (101) local to local area network 102 and a second redirection server R2 (103) local to local area network 104. Local area networks 102 and 104 are separated and connected over a wide area network (Internet 105). A plurality of clients, 106-1-106-N, are connected to local area network 102 and a plurality of clients, 107-1-107-M, are connected to local area network 104. All requests for web documents made by **clients** 106-1-106-N are passed through their local redirection **server** 101 and redirected to a caching **server** on system 100. Similarly, all requests for web documents made by **clients** 107-1-107-M are passed through their local redirection **server** 103 and redirected to a caching **server** on system 100. In the illustrative embodiment, a first caching server S1 (110) is connected local to local area network 102, a second caching server S2 (111) is connected to the wide area network Internet 105, and a third caching server S3 (112) is connected to local area network 104. In addition to caching individual web documents, each of these caching **servers** is responsible for particular hot sites, or web sites to which a large number of **client** requests are directed. Each caching server is responsible for one or more of such hot sites and each such hot site may be associated with more than one of the caching servers.

Detailed Description Text - DETX (4):

In the system 100, communication between **clients** and caching **servers** is effected over TCP/IP connections established over a network in a conventional manner. Each of the elements of system 100 may include a processor and a memory. The system 100 is suitable for implementing Hypertext Transfer Protocol (HTTP)-based network services on the Internet in a manner well known to one skilled in the art. For example, a client may generate an HTTP request for a particular web document on a hot site by designating a uniform resource locator (URL) of that document at the domain name of the hot site. Such a request is passed through the requesting **client's** local redirection **server** and redirected to one of the caching **servers,** S1, S2 or S3, which is responsible for that hot site. A TCP/IP connection is then established between the

requesting **client** and the particular caching **server** responsible for that hot site as selected by the local redirection **server**. If the particular requested document is available at that caching **server**, it is supplied to the requesting **client**. If not, a separate TCP/IP connection is established by the caching **server** to the actual hot site from where a copy of the requested web document is obtained and forwarded to the **cli nt**.

Detailed Description Text - DETX (5):
In the aforenoted co-pending patent application, a load distribution algorithm is presented for determining an initial distribution of a set of documents across servers and a determination of the probabilities, or weights, with which a redirector server should redirect requests to those particular servers that contain a replica of a requested document. Such load distribution is undertaken to **balance** the load on the set of servers where the definition of load **balance** is that the sum of access rates of requests to the documents redirected to each of the servers containing the documents is **balanced** across all the servers. Using as input the access rates of each document, the number of replicas of these documents that need to be made on the servers, and the capacity of each server, an initial distribution algorithm, described in the aforenoted co-pending application incorporated herein, produces a mapping between the documents and the servers as well as producing as output the probabilities (or weights) to be used by a redirecting server to redirect requests from/to replicated documents to one of the severs to which they are mapped to achieve the desired load **balance**.

Detailed Description Text - DETX (6):
FIG. 2 shows a flow network model in the co-pending patent application between a source and a sink for five documents, numbered 1-5 distributed on three servers, S1, S2 and S3, which each have equal scaled capacities of 0.333. The flow network is of a type described in, for example, R. K. Ahuja et al., "Network Flows: Theory, Algorithms and Applications", Prentice Hall, 1993, which is incorporated by reference herein. The scaled access rates to each of the documents, totaling 1, are shown on the five arcs between the source node and the redirector/document nodes. These access rates represent the probabilistic proportion of requests arriving at the redirector for each of the five documents. As can be noted from the arcs between the redirector/document nodes and the three server nodes, server S1 has stored replicas of document 1, 2, 3 and 4, server S2 has stored replicas of documents 1, 2, 4 and 5, and server S3 has stored replicas of documents 2, 3 and 5. These documents have been distributed in accordance with the initial distribution algorithm described in the aforenoted co-pending application to assure that at least two replicas of each document are stored in the set of three servers. The numbers on the arcs between the redirector/document nodes and the servers nodes represent the network flow solution for dividing the incoming access rate to the redirector/document nodes for each document to the servers that contain the document so that the total load to the servers S1, S2 and S3 is **balanced**. In the FIG. 2, the redundant arcs from the servers to the sink represent arcs that have infinite capacity but have a "high" cost associated with them. The cost on all the other arcs is equal to zero. The "high" cost arcs are used for overflows that may result when a change occurs in the system.

Detailed Description Text - DETX (7):
The distribution represented in FIG. 2 is a maximum-flow minimum-cost solution to the corresponding network flow problem that models the load **balancing** problem which must be solved to determine the desired solution. The solution is obtained using the mathematical language AMPL as described by R. Fourer et al., "AMPL: A Modeling Language for Mathematical Programming," The Scientific Press, 1993, which is incorporated by reference herein, in conjunction with a non-linear program solver such as MINOS. The numbers on the arcs between the redirector/document nodes and the server nodes represent the portions of the access rates for each document that are redirected to each server that contains the document. Thus, for example 0.175 of the 0.35 access rate requests for document 1 is redirected to server S1, and a similar portion is redirected to server S2. Thus, requests for document 1 received by the redirector should be redirected to servers S1 and S2 with equal weights or probabilities. Similarly, 0.113 of the 0.5 access rate for document 2 should be redirected to server S1, 0.108 of the 0.5 access rate should be redirected to server S2, and 0.279 of the 0.5 access rate should be redirected to server S3.

Detailed Description Text - DETX (9):
Although the above example and others in the co-pending patent application illustrate load **balancing** initial distributions involving a single redirector server which redirects requests to a plurality of document servers on which documents are replicated, the single redirector model can be extended to accommodate multiple redirectors. FIG. 3 models the same web server system as in FIG. 2, but includes two redirectors, as in the system of FIG. 1. In FIG. 3 redirector/document nodes 1 through 5 now represent requests for documents 1 through 5, respectively, which are redirected to the servers S1, S2 and S3 by redirector 1, and redirector/document nodes 1' through 5' represent requests for documents 1 through 5, respectively, which are redirected to the servers S1, S2 and S3 by redirector 2. The scaled access rate of each document is now, however, divided between the two redirectors. As an example, the scaled access rate for document 1, which in FIG. 2 is 0.35, is divided between the two redirectors so that the access rate to document 1 redirected through redirector 1 is 0.25 and the access rate redirected through redirector 2 is 0.1. The access rates for document 2 through redirectors 1 and 2 are 0.1 and 0.4, respectively; for document 3 through redirectors 1 and 2 are 0.04 and 0.01, respectively; for document 4 through redirectors 1 and 2 they are each 0.02; and for document 5 through redirects 1 and 2 they are 0.05 and 0.01, respectively. In the single redirector model of FIG. 2, equal proximity between the redirector and the servers was assumed so that costs on the arcs between the document nodes and the server nodes were not considered as variable parameters. It can be assumed, however, as shown in the network of FIG. 1, that the distances, d1j, 1.ltoreq.j.ltoreq.3, between redirector 1 and servers S1, S2 and S3 are such that d11<d 12<d13. Thus, as in the web network in FIG. 1, S1 is local to redirector 1 on local network 102, is separated from server S2 on the Internet 105 by a mid-range distance, and is furthest from S3 which is connected on a distant local network 104. Thus, for the two-redirector arrangement in FIG. 3, redirector 1 should redirect requests for a document to server S1 if it is available there and redirect requests for a document to server S2 and then S3 only if the capacity of S1 is exceeded. To

model this distance factor, costs are assigned to the arcs between the redirector/document nodes and the server nodes according to the distance between the redirector and the server. For example, requests for document 2 from redirector 1, if redirected to nearby server S1 are designated as having a cost per unit flow of 1, if redirected to mid-distance server S2 are designated as having a cost per unit flow of 2, and if redirected to distant server S3 are designated as having a cost per unit flow of 3. These same costs are assigned between the other redirector/document nodes associated with redirector 1 and the three servers. In a similar manner, redirector 2 is considered local to server S3, as shown in FIG. 1, is separated from S2 by a mid-range distance, and is separated from S1 by a long distance. In FIG. 3, the numerals in the square parentheses represent the costs on the arcs between the redirector/document nodes and the server nodes that contain replicas of the documents.

Detailed Description Text - DETX (10):
The cost on the overflow arcs from the servers to the sink needs to be carefully controlled. If load **balance** is the primary objective, then the costs on these arcs are chosen such that they are larger than the largest cost on the arcs that connect the redirector/document nodes to the server nodes. Otherwise, the network solution will lead to redirecting a flow to a close proximity device even if the load **balance** condition is violated as opposed to redirecting the flow to a device that is further away without violating the load **balance** requirement. For example, if the cost on the overflow arc from node S1 to the sink is chosen to be between 2 and 3, for all documents that are available in S1 and S3, redirector 1 will deterministically send the request to S1 possibly overflowing the capacity of server S1 even if server S3 has spare capacity. If the cost on the overflow arc is less than 2, requests for documents available on S1 and elsewhere will always be sent to S1 even if it overflows the capacity of Si. Where load **balance** is the primary objective, therefore, the costs on the overflow arcs are chosen to be 4, which is larger than the cost on any of the arcs between the redirector/document nodes and the server nodes. FIG. 3 shows the network flow solution, with flows between redirector/document nodes being specified without any parentheses around them, capacities specified within curved parentheses, and, as noted, costs specified within square parentheses. Where the capacity or the cost of an arc is not specified, its default value is 1. Using the determined flow values, the redirection probabilities, or weights, with which a request is forwarded to a particular server that has a stored copy of the requested document can be determined. Thus, for example, the flow on the arc from the redirector 1/document 1 node to S1 specifies that of the 0.25 units of scaled flow for the document 1/redirector 1 pair, 0.17 units should be redirected to server S1.

Detailed Description Text - DETX (12):
Unlike the solutions for load **balancing** focused on by the prior art, the present invention provides a solution for load sharing which considers the network delay associated with accessing a server and the server delay, which itself is a function of the access rate on that server. A network flow approach to this problem is considered after an initial distribution is completed for load **balance**. The aim of this load sharing solution is to minimize the average delay of all connections across all servers per unit of

time. FIG. 1 is the network model of a preferred embodiment of this invention. As previously noted, hot sites rather than individual documents are mapped to servers S1, S2 and S3, which, in this embodiment, are proxy cache devices. The network delay, which is the sum of the round-trip propagation and the transmission delays, is modeled by specifying them in the network flow model as costs on the arcs from the redirector/logical item nodes to the proxy cache server nodes, where the logical items are the different cached hot sites. The links from the proxy cache server nodes to the sink have a cost associated with them which is a function of the flow through these links. This cost models the device delay. The larger the number of connections a server serves per unit of time, the larger the device delay for each of the connection requests. The capacities of these links are not now used to specify a **balance** of load among the server devices, but to specify a limit on the average number of connections per time unit sent to the server device above which the connections may be dropped. As is described below, the proxy cache server delay and the capacity of the proxy cache server are calculated using a queuing model.

Detailed Description Text - DETX (15):
Using Little's Law (see L. Kleinrock, Queuing Systems, Vol. 1, Prentice Hall), the average **response time** or the device delay at the server can be computed as ##EQU3##

Detailed Description Text - DETX (19):
It can be noted that when K.fwdarw..infin., the **response time** approaches ##EQU6##

Detailed Description Text - DETX (20):
which is the **response time** of a M/M/1 system with infinite queue size.

Detailed Description Text - DETX (27):
FIG. 5 illustrates a network flow model 500 of the server system 100 in FIG. 1 in which the parameter values noted above have been incorporated. From the initial distribution, caching server S1 is responsible for hot sites 1, 2, 3 and 5; caching server S2 is responsible for hot sites 1, 3, 4 and 5; and caching server S3 is responsible for hot sites 2, 3 and 5. This mapping is noted in FIG. 1 within the S1, S2 and S3 caching servers. This is parallel to the mapping of documents on the three servers used in conjunction with the description of FIG. 2 for load **balancing**. From source 501, a total of 800 connection requests per second are generated for all hot sites. The requests flowing through redirector 101 are represented by the arcs between the source 501 and the redirector/hot site nodes 1-5 and the requests flowing through redirector 103 are represented by the arcs between the source 501 and the redirector/hot site nodes 1'-5'. The numbers associated with the arcs from source 501 to the redirector/hot site nodes 1-5, and 1'-5' specify the connection requests flowing through each respective redirector to each hot site. The cost representing the network delays are shown in the square brackets on the arcs between each redirector/hot site node and the server nodes. Each caching server is assumed, as noted above, to support up to 50 connections per time unit without dropping any connections. If more requests

are directed to these devices, then some connections may be dropped. All overflow connections are handled through the overflow arcs between the server nodes and the sink 502. The cost parameter chosen for these arcs (represented in FIG. 5 as infinity [.infin.]) should be chosen to be a larger number than the cost on all other arcs. The server delay at the proxy caches, shown as the cost on the arcs between each proxy cache server and the sink 502, is given by R(.lambda..sub.1), R(.lambda..sub.2), and R(.lambda..sub.3), where .lambda..sub.i is equal to the connection requests directed to caching server S1 and R(.lambda..sub.i) is calculated using equation (4).

Detailed Description Text - DETX (28):

The model in FIG. 5 is solved using the aforenoted AMPL mathematical programming language and the MINOS non-linear program solver with a goal of finding a solution to the non-linear problem such that the average **response time** for all of the 800 connections arriving per second is minimized. AMPL is a mathematical programming language that can be used to specify different optimization problems. It is used here to specify a minimum cost network flow optimization problem. For example, the load sharing network flow model of FIG. 5 can be modeled using the program shown in Appendix 1. The program defines the nodes in lines 101-103, the connectivity of the node in lines 104-106, specifies that the flow in must equal the flow out at line 107, specifies various criteria line 108-113, the function to be minimized in line 114 (postulated in the program as a minimization of total cost), and the different constraints of the problem in lines 115-118. The different parameters of this network, such as the number of hot sites, the number of redirectors and the number of proxy cache servers can be varied by specifying them in a data file with which the program is associated. Further, the input parameters of the model such as access rates, delays and capacities are also specified in this data file. The data file for the example in FIG. 5 is shown in Appendix 2. The AMPL environment takes the model file and data file and uses one of several different solvers to solve the optimization problem depending on the nature of the problem. As the specified problem is a non-linear optimization problem, the MINOS solver is used to solve the problem. The output of the solver provides the flow on each link between the redirector/hot-site nodes to the server. These flow values, noted on these arcs are then used by the redirectors to determine the probability or weights with which a request for a hot site is redirected by that redirector to a particular proxy cache server that is responsible for that requested hot site.

Detailed Description Text - DETX (30):

It can be noted from FIG. 5 that the load is now not **balanced** but shared among the servers such that the average delay is minimized. From the solution, the total minimized delay of all the 800 connections is calculated to be 15860 ms for an average delay of 19.825 ms per connection. The number of connections redirected to each device is noted to be less than or equal to the maximum number of connections that can be handled and thus the condition on the probability of dropped connections is satisfied. Thus, as can be noted in FIG. 5, there is zero flow in the overflow arcs between the server nodes and sink 502.

Detailed Description Text - DETX (32):
Access rate information to each hot site is determined by each **r director by associating the destination address** in the SYN packets with a set of hot site IP addresses. Alternatively, this information can be collected by examining, at the redirectors, the HOST field in the GET packets.

Detailed Description Text - DETX (33):
In the network configuration of FIG. 1 in which redirector 101 is local to clients 106-1-106-N and redirector 103 is local to clients 107-1-107-M, the redirectors do not have to treat traffic from different local clients in different manners. Network delay can therefore be accounted for by just considering the delay from a each redirector to each of the different caching servers. For a symmetric flow of traffic in which packets from the **clients** to the caching **servers** and from the caching **servers to the clients** flow through the redirector, network delay can be tracked by each redirector by computing the time between redirecting a SYN packet to a particular caching **server** and receiving the corresponding SYN ACK packet back from that **server**. In an asymmetric flow of traffic in which the **client**-to-**server** traffic flows through the redirector but the reverse traffic flows directly from the **server to the client,** the SYN ACK packet will not flow through the redirector. Therefore, network delay can be measured with another mechanism such as by PINGing the servers periodically.

Detailed Description Text - DETX (35):
Once an initial distribution of hot sites onto caching servers is performed based on access rate information to achieve load **balancing** in the manner specified in the co-pending application, CMS 115 performs a network flow computation for purposes of load sharing to determine the weights with which the redirectors should redirect requests to replicated hot site caches. Using these determined weights at each redirector minimizes the average delay of all connections across all of the caching servers per unit of time. This load sharing network flow computation is continually updated by periodically collecting current access rate and network delay information from each redirector and server delay information from the cache servers. Thus, the weights are continually updated based on the latest access rate, network delay and server delay information. Further, if the CMS 115 detects a failure of a caching server, it will trigger a network flow computation. In this case, the corresponding node is removed from the network flow model as well as all arcs incident upon it. Further, the service rate, .mu., may change at a caching server if, for example, one of two processors fails or if a device is replaced by another device with a higher performance CPU. This will affect two parameters: a) the delay at a server given that a specific number of connections are redirected to that server; and b) the capacity of the server in terms of the maximum flow that can be redirected to that server. As a result of a service rate change, a network flow computation can be triggered. Even further, if the access rate to a particular hot site suddenly and dramatically changes, a redirector will trigger a network flow computation. In this case, the flows on the arcs from the source to the redirector/hot site nodes are changed on the network flow model. Changes in other parameters can also affect the network flow computation. Thus, if the relative round-trip delay from the redirectors to the caching servers changes, the costs associated with the arcs

from the nodes representing the redirectors to the server nodes are changed. Also, since server load and server delay are determined by the number of requests redirected to a caching server, a change in such number of requests per second redirected to a server will change the server delay parameter.

Detailed Description Text - DETX (36):
FIG. 6 is a flowchart detailing the steps of the method of the present invention. At step 601, access rate information is obtained by CMS 115 from each redirector. At step 602, using this access rate information, an initial distribution of hot sites on the caching servers is determined using the prior art load **balancing** network flow algorithm. Once the initial distribution is determined, at step 603, CMS 115 obtains current access rate and network delay information for each redirector, and the server delay of each server is calculated. Using these inputs, at step 604, a network flow problem for load sharing is solved. At step 605, the probabilities (or weights) for each redirector for each hot site pair are determined and sent to each redirector. At decision step 606, a determination is made whether there has been an access rate change at a redirector. If yes, an update is triggered at step 607, which in turn causes the current access rate, network delay, and server delay to be determined or calculated back at step 603. Similarly, at decision step 608, a determination is made whether a server failure is detected. If yes, an update is triggered again at step 607. Further, at decision step 609, a determination is made whether or not a change in the delay at a caching server is detected. If yes, an update is triggered at step 607. If an access rate change, a server failure, or a caching server delay change are not detected at either decision steps 606, 608 or 609, respectively, then, at decision step 610, a determination is made whether the elapsed time since the last update has exceeded a threshold period of time, T. If not, the flow returns to the inputs of decision steps 606, 608 and 609. If the elapsed time has exceeded T, then an update is triggered at step 607.

Detailed Description Text - DETX (38):
Although described in conjunction with a system designed to achieve load **balance** across plural caching servers containing replicated hot sites for which in the network flow model the logical item at each redirector/logical item node represents a hot site, the present invention is not limited to such an arrangement. Thus, rather than having the logical items which are mapped to different servers being hot sites, as described above, the logical items could be any group of hot documents which are mapped onto a plurality of local caching servers in accordance with the document's origin server IP addresses. The origin server IP addresses of the documents are hashed to different groups. The different groups are then considered the logical items which are mapped onto the plural caching servers. Since many origin servers use multiple IP addresses, to avoid the same origin server name from being mapped to multiple caches, the hashing function is chosen so that the all IP addresses for an origin server are mapped to the same group out of a possible 256 different groups. Since origin servers normally use a contiguous block of IP addresses, then if the hashing is based on the first 8 bits of the origin server IP address, this contiguous block of IP address will be automatically mapped to the same group.

Detailed Description Text - DETX (41):

In the embodiments discussed hereinabove, it has been assumed that the clients are local to one of the redirectors. Therefore, the network delay between the client and the redirector has not been considered as a factor in the solution to the load sharing problem. The other possibility is for the redirectors to be closer to a plurality of servers. In this scenario, a **client's** request for a logical name is resolved by a Domain Name **Server into a redirector's IP address, which in turn redirects** the request to a **server** in a cluster of essentially duplicated **servers**. In this case, server side load **balancing** is achieved, in accordance with the present invention, so as to minimize the average delay per unit of time of all requests of all connections across all the servers in the cluster. In such an embodiment, **client IP addresses** are mapped into groups by a simple hashing function. As an example, the first 8 bits of the **client IP addresses** can be used to determine which of 256 client groups a client is mapped into. Other hashing functions could be found that evenly distribute the clients among the groups such that the access rate of clients allocated to each of the groups are evenly distributed among the groups. Regardless of the hashing function, each group becomes a logical item which is mapped onto the physical devices, which are, in this case, the back-end servers. The initial distribution algorithm can be used to map these logical items to the servers. The network flow based algorithm can then be used to compute the redirector probabilities associated with each redirector/logical item pair, which in this case is a redirector/client group pair. In the network flow model then, the arcs between the source and each redirector/**client** group pair represent the requests generated from each group that need to be redirected to one of the back-end **servers**. The load sharing solution to the network flow problem, in accordance with the present invention, will produce the weights with which the redirectors direct requests from each of the **client** groups to the plural **servers**. The cost associated with each arc represents the delay. Thus, the cost on an arc between the source and one of the redirector/**client** group nodes represents the network delay encountered between a **client** group and the redirector, while the cost on an arc between a redirector/**client** group node and a **server** represents network delay between the redirector and one of the **servers**, plus that **server's** delay. In the hot site embodiment of the present invention previously described, it was assumed that the clients and the redirectors were local to one another so that the delay between each was a constant small value, and thus not a variable. With server side load sharing, this delay is certainly a variable that is considered in the network flow load sharing solution.

Detailed Description Text - DETX (42):

If the redirector is local to the back-end servers, then the network delay computation does not have to include the delay from the redirector to the servers. By grouping the clients together in groups according to their IP addresses, the clients within each group are likely to be geographically proximate. Thus, the network delay can be determined for each group of **clients** and a specific **server** by determining the network delay from each **client** group to the redirector. For a symmetric traffic flow, the delay from the **client** group to the redirector can be calculated at the redirector by keeping track of the time between redirecting a SYN ACK packet from the **server to the client** group and the time an ACK packet for the SYN ACK packet is received at the

redirector as part of the three-way TCP handshake. With an asymmetric traffic flow model, the SYN ACK packet does not flow through the redirector. In this case, the redirector can periodically PING one or more clients in a client group to calculate the delay from that client group to the redirector. Further, the redirector separately keeps track of the access rates from each client group in order to solve the network flow for load sharing.

Detailed Description Text - DETX (43):

If the redirector, rather than being local to the back-end **servers,** functions as a virtual **server** which, upon receipt of a request, redirects the request to one of a plurality of **servers** located in different locations on the wide area network, then the delay between the redirector and each **server** also needs to be taken into account in addition to the delay between each **client** group and the redirector. If the traffic flow is symmetric, the redirector can calculate the network delay from a **client** group to a **server** by adding the delay from the **client** group to the redirector to the delay from the redirector to the **server**. The delay from the redirector to the server can be calculated as previously described by keeping track of the time between redirecting a SYN packet to a server and receiving the corresponding SYN ACK packet. As before, the delay from the **client** group to the redirector can be calculated at the redirector by keeping track of the time between redirecting the SYN ACK packet from the **server to the client** group and the time an ACK packet for this SYN ACK packet is received at the redirector. If the traffic flow is asymmetric, the delay on the traffic that flows directly from the **server to the client** is of interest. The **client to server** delay is not as an important parameter since most of the HTTP traffic flows from the **servers to the clients**. The mechanism used for the symmetric flow case can be used as an approximation for the **server**-to-**client** delay or it can be estimated by the redirector using geographical information, such as that provided by IANA as used in Classless Inter-Domain Routing Protocol.

Detailed Description Text - DETX (44):

The network flow model can thus be used to solve the load sharing problem when the logical item in the redirector/logical item pair is associated with the **clients** making requests to a **server or the servers** providing responses to requests from **clients**. Further, the network model is flexible to provide a solution when a delay is associated with the link between a **client** group and the redirector, on the link between the redirector and the **server,** or on both links.

Claims Text - CLTX (1):

1. A method of processing **client** requests through at least one redirector to a plurality of **servers** connected on a communications network to minimize an average delay associated with the **client** requests, at least some of the **client** requests being capable of being satisfied by more than one of the **servers,** the method comprising the steps of:

Claims Text - CLTX (3):

b) determining a network delay between each of a plurality of **clients** and

the plurality of **servers**;

Claims Text - CLTX (4):
c) determining a **server** delay incurred in processing a **client** request at each of the plurality of **servers**;

Claims Text - CLTX (5):
d) using the determined access rates of requests in step a), the network delays determined in step b) and the **server** delays determined in step c) as inputs, solving a non-linear program optimization problem to determine a set of weights associated with each of the plurality of redirector-logical item pairs so as to minimize the average delay associated with the **client** requests; and

Claims Text - CLTX (6):
e) probabilistically forwarding a **client** request through the at least one redirector to a **server** that can satisfy that request using the determined weights associated with the redirector-logical pair item.

Claims Text - CLTX (27):
22. In a system which processes **client** requests through at least one redirector to a plurality of **servers** connected on a communications network, at least some of the **client** requests being capable of being satisfied by more than one of the **servers**, apparatus for minimizing an average delay associated with the **client** requests, the apparatus comprising:

Claims Text - CLTX (29):
means for determining a network delay between each of a plurality of **clients** and the plurality of **servers**;

Claims Text - CLTX (30):
means for determining a **server** delay incurred in processing a **client** request at each of the plurality of **servers**;

Claims Text - CLTX (31):
means for solving a non-linear programming optimization problem to determine a set of weights associated with each of the plurality of redirector-logical items pairs so as to minimize the average delay of **client** requests using the determined access rate of requests, the determined network delays, and the determined **server** delays as inputs to the problem; and

Claims Text - CLTX (32):
means for probabilistically forwarding a **client** request through the at least one redirector to a **server** in the system that can satisfy that request using the determined weights associated with the redirector-logical pair item.

Claims Text - CLTX (53):
43. In a system which processes **client** requests through at least one redirector to a plurality of **servers** connected on a communications network, at least some of the **client** requests being capable of being satisfied by more than one of the **servers,** a method of determining a set of weights with which the at least one redirector will probabilistically forward **client** requests to the **server** in the system that can satisfy the requests comprising the steps of:

Claims Text - CLTX (55):
b) determining a network delay between each of a plurality of **clients** and the plurality of **servers**;

Claims Text - CLTX (56):
c) determining a **server** delay incurred in processing a **client** request at each of the plurality of **servers**; and

Claims Text - CLTX (57):
d) using the determined access rates of requests in step a), the network delays determined in step b) and the **server** delays determined in step c) as inputs, solving a non-linear program optimization problem to determine the set of weights associated with each of the plurality of redirector-logical item pairs so as to minimize the average delay associated with the **client** requests.

Claims Text - CLTX (78):
64. A system for processing **client** requests to a plurality of **servers** connected on a communications network, at least some of the **client** requests being capable of being satisfied by more than one of the **servers,** the system comprising:

Claims Text - CLTX (82):
b) determining a network delay between each of a plurality of **clients** and the plurality of **servers**;

Claims Text - CLTX (83):
c) determining a **server** delay incurred in processing a **client** request at each of the plurality of **servers**; and

Claims Text - CLTX (84):
d) using the determined access rates of requests in step a), the network delays determined in step b) and the **server** delays determined in step c) as inputs, solving a non-linear program optimization problem to determine a set of weights associated with each of the plurality of redirector-logical item pairs so as to minimize the average delay associated with the **client** requests;

Claims Text - CLTX (85):
the at least one redirector using the determined weights associated with each redirector-logical item pair to probabilistically forward each **client** request to a **server** that can satisfy that request.

Other Reference Publication - OREF (4):
Azar Bestavros., "WWW Traffic Reduction and Load **Balancing** through Server-Based Caching". 1063-6552/97 IEEE 1997 pp. 56-67.*

US-PAT-NO: 6665702

DOCUMENT-IDENTIFIER: US 6665702 B1

TITLE: Load **balancing**

---------- KWIC ---------

TITLE - TI (1):
Load **balancing**

Parent Case Text - PCTX (2):
This application is a continuation-in-part of assignee's application U.S. Ser. No. 09/115,643, filed on Jul. 15, 1998, and entitled "Load **Balancing,**" now U.S. Pat. No. 6,249,801.

Brief Summary Text - BSTX (2):
The present invention relates to computer networks in general, and in particular to load **balancing client** requests among redundant network **servers** in different geographical locations.

Brief Summary Text - BSTX (4):
In computer networks, such as the Internet, preventing a **server** from becoming overloaded with requests from clients may be accomplished by providing several **servers** having redundant capabilities and managing the distribution of client requests among the **servers** through a process known as "load balancing."

Brief Summary Text - BSTX (5):
In one early implementation of load **balancing,** a Domain Naming System (DNS) server connected to the Internet is configured to maintain several IP addresses for a single domain name, with each address corresponding to one of several servers having redundant capabilities. The DNS **server** receives a request for address translation and responds by returning the list of **server** addresses from which the client chooses one address at random to connect to. Alternatively, the DNS server returns a single address chosen either at random or in a round-robin fashion, or actively monitors each of the servers and returns a single address based on server load and availability.

Brief Summary Text - BSTX (6):
More recently, a device known as a "load **balancer,**" such as the Web Server Director, commercially available from the Applicant/assignee, has been used to **balance** server loads as follows. The load **balancer** is provided as a gateway to several redundant servers typically situated in a single geographical location and referred to as a "server farm" or "server cluster." DNS servers store the

IP address of the load **balancer** rather than the addresses of the servers to which the load **balancer** is connected. The load **balancer's** address is referred to as a "virtual IP address" in that it masks the addresses of the servers to which it is connected. Client requests are addressed to the virtual IP address of the load **balancer** which then sends the request to a **server** based on **serv r** load and availability or using other known techniques.

Brief Summary Text - BSTX (7):
Just as redundant **servers** in combination with a load **balancer** may be used to prevent **server** overload, redundant **server** farms may be used to reroute client requests received at a first load balancer/server farm to a second load balancer/server farm where none of the **servers** in the first **server** firm are available to tend to the request. One rerouting method currently being used involves sending an HTTP redirect message from the first load balancer/server farm to the client instructing the client to reroute the request to the second load balancer/server farm indicated in the redirect message. This method of load **balancing** is disadvantageous in that it can only be employed in response to HTTP requests, and not for other types of requests such as FTP requests. Another rerouting method involves configuring the first load **balancer** to act as a DNS server. Upon receiving a DNS request, the first load **balancer** simply returns the virtual IP address of the second load **balancer**. This method of load **balancing** is disadvantageous in that it can only be employed in response to DNS requests where there is no guarantee that the request will come to the first load **balancer** since the request does not come directly from the client, and where subsequent requests to intermediate DNS **servers** may result in a previously cached response being returned with a virtual IP address of a load **balancer** that is no longer available.

Brief Summary Text - BSTX (8):
Where redundant **server** farms are situated in more than one geographical location, the geographical location of a client may be considered when determining the load **balancer to which the client's** requests should be routed, in addition to employing conventional load **balancing** techniques. However, routing client requests to the geographically nearest **server,** load **balancer, or server** farm might not necessarily provide the client with the best service if, for example, routing the request to a geographically more distant location would otherwise result in reduced latency, fewer hops, or provide more processing capacity at the **server**.

Brief Summary Text - BSTX (10):
The present invention seeks to provide novel apparatus and methods for load **balancing client** requests among redundant network **servers and server** farms in different geographical locations which overcome the known disadvantages of the prior art as discussed above.

Brief Summary Text - BSTX (11):
There is thus provided in accordance with a preferred embodiment of the present invention a method for load **balancing** requests on a network, the method including receiving a request from a requestor having a requestor network

address at a first load **balancer** having a first load **balancer** network address, the request having a source address indicating the requestor network address and a destination address indicating the first load **balancer** network address, forwarding the request from the first load **balancer** to a second load **balancer** at a triangulation network address, the request source address indicating the requestor network address and the destination address indicating the triangulation network address, the triangulation network address being associated with the first load **balancer** network address, and sending a response from the second load **balancer** to the requester at the requestor network address, the response having a source address indicating the first load **balancer** network address associated with the triangulation network address and a destination address indicating the first requestor network address.

Brief Summary Text - BSTX (12):
Further in accordance with a preferred embodiment of the present invention the method includes maintaining the association between the triangulation network address and the first load **balancer** network address at either of the load balancers.

Brief Summary Text - BSTX (13):
Still her in accordance with a preferred embodiment of the present invention the method includes maintaining the association between the triangulation network address and the first load **balancer** network address at the second load **balancer**, and communicating the association to the first load **balancer**.

Brief Summary Text - BSTX (14):
Additionally in accordance with a preferred embodiment of the present invention the method includes directing the request from the second load **balancer** to a server in communication with the second load **balancer**, composing the response at the server, and providing the response to the second load **balancer**.

Brief Summary Text - BSTX (15):
There is also provided in accordance with a preferred embodiment of the present invention a method for load **balancing** requests on a network, the method including determining the network proximity of a requestor with respect to each of at least two load balancers, designating a closest one of the load balancers by ranking the load balancers by network proximity, and directing requests from the requestor to the closest load **balancer**.

Brief Summary Text - BSTX (16):
Further in accordance with a preferred embodiment of the present invention the method includes directing requests from any source having a subnet that is the same as the subnet of the requester to the closest load **balancer**.

Brief Summary Text - BSTX (17):
Still further in accordance with a preferred embodiment of the present

invention the method includes monitoring the current load of each of the load balancers, and performing the directing step the current load of the closest load **balancer** is less than the curt load o every other of the load balancers.

Brief Summary Text - BSTX (23):

There is also provided in accordance with a preferred embodiment of the present invention a method for determining network proximity, the method including sending from each of at least two **servers** a UDP request having a starting TTL value to a client at a sufficiently high port number as to elicit an "ICMP port unreachable" reply message to at least one determining one of the **servers** indicating the UDP request's TTL value on arrival at the client, determining a number of hops from each of the **servers to the client** by subtract the starting TTL value from the TTL value on arrival for each of the **servers,** and determining which of the **servers** has fewer hops of the client, and designating the **server** having fewer hops as being closer to the **client than the other of the servers**.

Brief Summary Text - BSTX (24):

There is additionally provided in accordance with a preferred embodiment of the present invention a network load **balancing** system including a network, a first load **balancer** connected to the network and having a first load **balancer** network address, a second load **balancer** connected to the network and having a triangulation network address, the triangulation network address being associated with the first load **balancer** network address, and a requestor connected to the network and having a requestor network address, where the requestor is operative to send a request via the network to the first load **balancer,** the request having a source address indicating the requestor network address and a destination address indicating the first load **balancer** network address, the first load **balancer** is operative to forward the request to the second load **balancer** at the triangulation network address, the request source address indicating the requestor network address and the destination address indicating the triangulation network address, and the second load **balancer** is operative to send a response to the requestor at the requester network address, the response having a source address indicating the first load **balancer** network address associated with the triangulation network address and a destination address indicating the first requestor network address.

Brief Summary Text - BSTX (25):

Further in accordance with a preferred embodiment of the present invention either of the load balancers is operative to maintain a table of the association between the triangulation network address and the first load **balancer** network address.

Brief Summary Text - BSTX (26):

Still further in accordance with a preferred embodiment of the present invention the second load **balancer** is operative to maintain a table of the association between the triangulation network address and the first load bar network address and communicate the association to the first load **balancer**.

Brief Summary Text - BSTX (27):
Additionally in accordance with a preferred embodiment of the present invention the system further includes a server in communication with the second load **balancer**, where the second load **balancer** is operative to direct the request from the second load **balancer** to the server, and the server is operative to compose the response and provide the response to the second load **balancer**.

Brief Summary Text - BSTX (28):
There is also provided in accordance with a preferred embodiment of the present invention a network load **balancing** system including a network, at least two load balancers connected to the network, and a requestor connected to the network, where each of the at least two load balancers is operative to determine the network proximity of the requestor, and at least one of the load balancers is operative to designate a closest one of the load balancers by ranking the load balancers by network proximity and direct requests from either of the requester and a subnet of the requester to the closest load **balancer**.

Brief Summary Text - BSTX (32):
It is noted that throughout the specification and claims the term "network proximity" refers to the quality of the relationship between a client and a first **server or server** farm as compared with the relationship between the client and a second **server or server** farm when collectively considering multiple measurable factors such as latency, hops, and **server** processing capacity.

Brief Summary Text - BSTX (41):
There is also provided in accordance with yet another preferred embodiment of the present a method for managing a computer network connected to the Internet through a plurality of ISPs, includes the steps of: receiving a request from a client within a computer network directed to a remote **server** computer, looking up a table entry within a proximity table indexed by an address related to the remote **server** computer, the tables entries of the proximity table containing ratings for a plurality of ISPs, and selecting one of the plurality of ISPs through which to route the client request, based on the ratings within the table entry looked up in the proximity table.

Brief Summary Text - BSTX (47):
The computer network may further be a private network, visible externally through a network address translation. Preferably the method may also include the steps of receiving a response from the remote **server** directed to the source IP address designated for the client request, and translating the source IP address designated for the client address to the IP address for the client within the private network.

Brief Summary Text - BSTX (54):
There is also provided in accordance with another preferred embodiment of

the present invention, a network management system for managing a computer network connected to the Internet through a plurality of ISPs, including a network controller receiving a client request from within a computer network directed to a remote **server** computer, and selecting one of a plurality of ISPs through which to route the client request, and a data manager looking up a table entry within a proximity table indexed by an address related to the remote **server** computer, the tables entries of the proximity table containing ratings for a plurality of ISPs. The network controller may also select one of the plurality of ISP based on the rating within the table entry looked up in the proximity table.

Brief Summary Text - BSTX (58):
Moreover in accordance with a preferred embodiment of the present invention, the network controller routes the client request through the selected ISP. Preferably the computer network is a private network, visible externally through a network address translation, and the network controller receives a response from the remote **server** directed to the source IP address designated for the client request, the system further comprising a network address translator translating the source IP address designated for the client address to the IP address for the client within the private network.

Brief Summary Text - BSTX (99):
Additionally the route selector is operable to configure and use a Decision Parameter Table comprising parameters of the routes. Furthermore, different Decision Parameters are supplied for each respective content type. The Decision Parameter Table also includes at least one of a group of parameter weights comprising; Data packet content; Hops weighting factor, Packet loss factor and **Response time** factor. It is appreciated that a different Decision Parameters is used for each respective content.

Brief Summary Text - BSTX (100):
A Decision Function F.sub.content is calculated for each path from the first node to the second node, based on said Decision Parameter Table. The Decision Function F.sub.content is defined as: F.sub.content =F(Hops weighting factor*Hops count factor; **Response weighting factor*Response time** factor; Path quality weighting factor*Path quality factor, Packet loss weighting factor*Packet loss factor).

Drawing Description Text - DRTX (3):
FIGS. 1A-1C, taken together, are simplified pictorial flow illustrations of a triangulation load **balancing** system constructed and operative in accordance with a preferred embodiment of the present invention;

Drawing Description Text - DRTX (4):
FIGS. 2A-2F, taken together, are simplified pictorial flow illustrations of a network proximity load **balancing** system constructed and operative in accordance with another preferred embodiment of the present invention;

Drawing Description Text - DRTX (5):
FIGS. 3A-3F, taken together, are simplified pictorial flow illustrations of a preferred embodiment of the present invention for managing and load **balancing** a multi-homed network architecture whereby a client is connected to the Internet through multiple ISPs; and

Detailed Description Text - DETX (2):
Reference is now made to FIGS. 1A-1C which, taken together, are simplified pictorial flow illustrations of a triangulation load **balancing** system constructed and operative in accordance with a preferred embodiment of the present invention. Two server farms, generally designated 10 and 12 respectively, are shown connected to a network 14, such as the Internet, although it is appreciated that more than two server farms may be provided. Server farms 10 and 12 typically comprise a load **balancer** 16 and 18 respectively, which may be a dedicated load **balancer** or a server or router configured to operate as a load **balancer,** with each of the load balancers being connected to one or more servers 20. Load balancers 16 and 18 are alternatively referred to herein as LB1 and LB2 respectively. LB1 and LB2 typically maintain a server status table 22 and 24 respectively, indicating the current load, configuration, availability, and other server information as is common to load balancers. LB1 and LB2 also typically periodically receive and maintain each other's overall status and load statistics such that LB1 and LB2 can know each other's availability.

Detailed Description Text - DETX (3):
Typical operation of the triangulation load **balancing** system of FIGS. 1A-1C is now described by way of example. As is shown more particularly with reference to FIG. 1A, a client 26, such as any known computer terminal configured for communication via network 14, is shown sending a request 28, such as an FTP or HTTP request, to LB1 whose virtual IP address is 100.100.1.0. In accordance with network transmission protocols, request 28 indicates the source IP address of the requestor, being the IP address 197.1.33.5 of client 26, and the destination IP address, being the virtual IP address 100.100.1.0 of LB1. LB2 preferably periodically sends a status report 30 to LB1, the virtual IP address 100.100.1.0 of LB1 being known in advance to LB2. Status report 30 typically indicates the availability of server farm 12 and provides load statistics, which LB1 maintains.

Detailed Description Text - DETX (4):
LB2 is preferably capable of having multiple virtual IP addresses as is well known. It is a particular feature of the present invention for LB2 to designate a currently unused virtual IP address, such as 200.100.1.1, for LB1's use and store the mapping between the IP address of LB1 and the designated IP address in a triangulation mapping table 32, as is shown more particularly with reference to FIG. 1B. The designated address is referred to herein as the triangulation address and may be preconfigured with LB1 or periodically provided to LB1 from LB2. LB1 preferably maintains in a client mapping table 36 a mapping of the IP address 197.1.33.5 of client 26 and the triangulation **address 200.100.1.1 of LB2 to which client 26's requests may be redirected**.

Detailed Description Text - DETX (5):

As shown in the example of FIG. 1A, **serv r** status table 22 of LB1 indicates that no **servers in server** farm 10 are available to service client 26's request, but indicates that **server** farm 12 is available. Having decided that client 26's request should be forwarded to LB2, in FIG. 1C LB1 substitutes the destination IP address of request 28 with the virtual IP address 200.100.1.1 of LB2 which is now mapped to the IP address of client 26 as per client mapping table 36 and sends an address-modified client request 38 to LB2. LB2, upon receiving request 38 at its virtual IP address 200.100.1.1, checks triangulation mapping table 32 and finds that virtual IP address 200.100.1.1 has been designated for LB1's use. LB2 therefore uses the virtual IP address 100.100.1.0 of LB1 as per triangulation mapping table 32 as the source IP address of an outgoing response 40 that LB2 sends to client 26 after the request has been serviced by one of the **servers in server** arm 12 selected by LB2. It is appreciated that response 40 must appear to client 26 to come from LB1, otherwise client 26 will simply ignore response 40 as an unsolicited packet. Client 26 may continue to send requests to LB1 which LB1 then forwards requests to LB2 at the designated triangulation address. LB2 directs requests to an available **server** and sends responses to client 26 indicating LB1 as the source IP address.

Detailed Description Text - DETX (6):

Reference is now made to FIGS. 2A-2F which, taken together, are simplified pictorial flow illustrations of a network proximity load **balancing** system constructed and operative in accordance with another preferred embodiment of the present invention. The configuration of the system of FIGS. 2A-2F is substantially similar to FIGS. 1A-1C except as otherwise described hereinbelow. For illustration purposes, a third server farm, generally designated 50, is shown connected to network 14, although it is appreciated that two or more server farms may be provided. Server farm 50 typically comprises a load **balancer** 52, which may be a dedicated load **balancer** or a server or router configured to operate as a load **balancer,** with load **balancer** 52 being connected to two or more servers 20. Load **balancer** 52 is alternatively referred to herein as LB3.

Detailed Description Text - DETX (7):

Typical operation of the network proximity load **balancing** system of FIGS. 2A-2F is now described by way of example. As is shown more particularly with reference to FIG. 2A, client 26 is shown sending request 28, such as an FTP or HTTP request to LB1 whose virtual IP address is 100.100.1.0. LB1 preferably maintains a proximity table 54 indicating subnets and the best server farm site or sites to which requests from a particular subnet should be routed. Determining the "best" site is described in greater detail hereinbelow.

Detailed Description Text - DETX (8):

Upon receiving a request, LB1 may decide to service the request or not based on normal load **balancing** considerations. In any case, LB1 may check proximity table 54 for an entry indicating the subnet corresponding to the subnet of the

source IP address of the incoming request. As is shown more particularly with reference to FIG. 2B, if no corresponding entry is found in proximity table 54, LB1 may send a proximity request 56 to LB2, and LB3, whose virtual IP addresses are known in advance to LB1. Proximity request 56 indicates the IP address of client 26.

Detailed Description Text - DETX (9):
A "network proximity" may be determined for a requester such as client 26 with respect to each load balancer/server farm by measuring and collectively considering various attributes of the relationship such as latency, hops between client 26 and each **server** farm, and the processing capacity and quality of each **server** farm site. To determine comparative network proximity, LB1, LB2, and LB3 preferably each send a polling request 58 to client 26 using known polling mechanisms. While known polling mechanisms included pinging client 26, sending a TCP ACK message to client 26 may be used where pinging would otherwise fail due to an intervening firewall or NAT device filtering out a polling message. A TCP ACK may be sent to the client's source IP address and port. If the client's request was via a UDP connection, a TCP ACK to the client's source IP address and port 80 may be used. One or both TCP ACK messages should bypass any intervening NAT or firewall and cause client 26 to send a TCP RST message, which may be used to determine both latency and TTL. While TTL does not necessarily indicate the number of hops from the client to the load **balancer,** comparing TTL values from LB1, LB2, and LB3 should indicate whether it took relatively more or less hops.

Detailed Description Text - DETX (11):
Client 26 is shown in FIG. 2D sending a polling response 60 to the various polling requests. The responses may be used to determine the latency of the transmission, as well as the TTL value. LB2 and LB3 then send polling results 62 to LB1, as shown in FIG. 2E. The polling results may then be compared, and LB1, LB2, and LB3 ranked, such as by weighting each attribute and determining a total weighted value for each server farm. Polling results may be considered together with **server** farm capacity and availability, such as may be requested and provided using known load **balancing** reporting techniques or as described hereinabove with reference to FIGS. 1A and 1B, to determine the **server** farm site that is "closest" to client 26 and, by extension, the client's subnet, which, in the example shown, is determined to be LB2. For example, the closest site may be that which has the lowest total weighted value for all polling, load, and capacity results. LB1 may then store the closest site to the client/subnet in proximity table 54.

Detailed Description Text - DETX (12):
As was described above, a load **balancer** that receives a request from a client may check proximity table 54 for an entry indicating the subnet corresponding to the subnet of the source IP address of the incoming request. Thus, if a corresponding entry is found in proximity table 54, the request is simply routed to the location having the best network proximity. Although the location having the best network proximity to a particular subnet may have already been determined, the load **balancer** may nevertheless decide to forward an incoming request to a location that does not have the best network proximity

should a load report received from the best location indicate that the location is too busy to receive requests. In addition, the best network proximity to a particular subnet may be periodically redetermined, such as at fixed times or after a predetermined amount of time has elapsed from the time the last determination was made.

Detailed Description Text - DETX (15):
Reference is now made to FIGS. 3A-3F, which illustrate a preferred embodiment of the present invention for managing and load **balancing** a multi-homed network architecture whereby a client is connected to the Internet through multiple ISPs. As illustrated in FIG. 3A, a client 105 is connected to the Internet 110 through three ISPs, 115, 120 and 125, each having a respective router 130, 135 and 140 to controls the flow of data packets. The system includes a content router 145, operative in accordance with a preferred embodiment of the present invention, to provide efficient connectivity between client 105 and Internet **servers, such as server** 150. As illustrated in FIG. 3A, client 105 has an IP address of 10.1.1.1 on a private network, and seeks to connect to **server** 150 having an IP address of 192.115.90.1.

Detailed Description Text - DETX (16):
As illustrated in FIG. 3B, ISPs 115, 120 and 125 assign respective IP address ranges to the client network, indicated in FIG. 3B by ranges 20.x.x.x, 30.x.x.x and 40.x.x.x. The first time that client 105 connects to **server** 150, content router 145 preferably sends polling requests through each of routers 130, 135 and 140 in order to determine the proximity of **server** 150 to client 105. When sending the polling requests, content router 145 assigns respective network addresses 20.1.1.1, 30.1.1.1 and 40.1.1.1 to client 105. Thus three polling requests are sent: one from each of the sources 20.1.1.1, 30.1.1.1 and 40.1.1.1 to destination 192.115.90.1.

Detailed Description Text - DETX (18):
Based on these polling results, content router 145 chooses, for example, router 135 as its first choice for connecting client 105 with **server** 150. As illustrated in FIG. 3D, proximity results are stored in a proximity table 155. Specifically, proximity table 155 indicates that router 135 is the first choice for connecting content router 145 to any computer residing on subset 192.115.90. Thus, when a new client 160 with IP address 10.2.2.2 on the private network attempts to connect to a **server** 165 with IP address 192.115.90.2, through a content router 145, content router 145 determines from proximity table 155 that the best router to use is router 135.

Detailed Description Text - DETX (20):
As illustrated in FIG. 3F, this ensures that subsequent responses sent back from server 165 will be addressed to IP address 30.1.1.1 and, accordingly, will be routed through ISP 120. Content router 145 in turn uses network address translation (NAT) data to determine that IP address 30.1.1.1 corresponds to private IP address 10.2.2.2, and transmits the responses from **server** 165 back to client 160.

Detailed Description Text - DETX (22):
FIG. 4B indicates a NAT mapping table 180, showing that the private IP address 10.3.3.3 for server 170 is translated to IP addresses 20.3.3.3, 30.3.3.3 and 40.3.3.3, respectively, by routers 130, 135 and 140. Content router 145 looks up the subnet entry 192.115.90 in proximity table 155, and identifies router 135 as the first choice for best proximity between **server** 170 and client 175. In resolving the DNS request, content router 145 accordingly provides 30.3.3.3 as the IP address for the domain name server, even though the original request indicated a destination IP address of 20.1.1.1. This ensures that requests from client 175 are sent to **server** 170 with a destination IP address of 30.3.3.3, which in turn ensures that the client requests are transmitted through ISP 120.

Detailed Description Text - DETX (23):
It can be seen from FIGS. 3A-3F that the present invention efficiently **balances** the load among the three ISPs 115, 120 and 125 for outgoing connections. Similarly, it can be seen from FIGS. 4A and 4B that the present invention efficiently **balances** the load among the three ISPs 115, 120 and 125 for incoming connections. In the event that the router indicated as first choice for the best proximity connection is unavailable or overloaded, the present invention preferably uses a second choice router instead. Thus the present invention ensures that if an ISP service is unavailable, connectivity to the Internet is nevertheless maintained.

Detailed Description Text - DETX (24):
Referring back to FIG. 3F, suppose for example that ISP 120 is unavailable, and that content router 145 routes the outgoing client request through ISP 125 instead of through ISP 120. In accordance with a preferred embodiment of the present invention, content router 145 routes the outgoing request through ISP 125 and labels the outgoing request with a source IP address of 40.1.1.1. Had content router 145 used ISP 125 but indicated a source IP address of 30.1.1.1, the response from **server** 150 would be directed back through ISP 125, and not be able to get through to client 160.

Detailed Description Text - DETX (29):
It is appreciated that the managing of the routing, by the content router 508, typically depends on the following factors: the content type, the number of hops to the destination, the **response time** of the destination, the availability of the path, the costing of the link and the average packet loss in the link.

Detailed Description Text - DETX (30):
In order for the content router 508 to determine the "best" path, a "Decision Parameter Table" is built for each content type. It is appreciated that the content type may vary between the application type and actual content (URL requested, or any other attribute in the packet). The Decision Parameter Table is preferably dependent on the parameters: Data packet content; Hops weighting factor; Packet loss factor and **Response time** factor. Typical values

of these parameters are also given in Table 1.

Detailed Description Text - DETX (31):
In addition to the parameters listed in Table 1, the following additional parameters may also be taken into consideration Hops count factor; **Respons time** factor; Path quality factor; and Packet loss factor.

Detailed Description Text - DETX (32):
A Destination Table is built to summarize the following factors: the content type, the number of hops to the destination, the **response time** of the destination, the availability of the path, and the average packet loss in the link, based on proximity calculations, as previously defined.

Detailed Description Text - DETX (33):
Using the relevant data, as typically listed in Table 1, the content router 508 determines a Decision Function F.sub.content for each path: F.sub.content =F(Hops weighting actor*Hops count factor, **Response weighting factor*Response time** factor, Path quality weighting factor*Path quality factor, Packet loss weighting factor*Packet loss factor).

Detailed Description Text - DETX (35):
Based on the Decision Function the content router 508 selects one of the available pats. The data packet is then routed through the selected path. The Decision Function for a particular path is determined by an administrative manager (not shown) and may depend, for example, on the minimum number of hops or on the relevant **response time,** or on the packet loss, or on the path quality, or any combination of the above parameters, according to the administrative preferences.

Detailed Description Text - DETX (38):
Thus it may be appreciated that the present invention enables a multi-homed network architecture to realize the full benefits of its redundant route connections by maintaining fault tolerance and by **balancing** the load among these connections, and preferably using data packet content information in an intelligent decision making process.

Claims Text - CLTX (1):
1. A method for managing a computer network connected to the Internet through a plurality of routes, comprising the steps of: receiving a request from a client within a client computer network directed to a remote **server** computer within a second computer network; looking up a table entry within a proximity table indexed by an address related to the remote **server** computer, the tables entries of the proximity table containing ratings for a plurality of routes between the client computer network and the second computer network; and selecting one of the plurality of routes through which to route the client request, based on the ratings within the table entry looked up in the proximity tables, wherein the plurality of routes assign respective IP addresses to the

computer network, and wherein the method further comprises the step of setting the source IP address of the client request corresponding to the selected route on the client side.

Claims Text - CLTX (7):

7. The method of claim 6 further comprising the steps of: receiving a response from the remote **server** directed to the source IP address designated for the client request; and translating the source IP address designated for the client request to the IP address for the client within the private network.

Claims Text - CLTX (8):

8. A network management system for managing a computer network connected to the Internet through a plurality of routes, comprising: a network controller receiving a client request from within a client computer network directed to a remote **server** computers, within a second computer network and selecting one of a plurality of routes through which to route the client request; and a data manager looking up a table entry within a proximity table indexed by an address related to the remote **server** computer, the tables entries of the proximity table containing ratings for a plurality of routes, between the client computer network and the second computer network and wherein said network controller selects one of the plurality of routes based on the ratings within the table entry looked up in the proximity tables, wherein the plurality of routes assign respective IP addresses to the computer network, and wherein said network controller sets the source IP address of the client request corresponding to the selected route on the client side.

Claims Text - CLTX (14):

14. The network management system of claim 13 wherein said network controller receives a response from the remote **server** directed to the source IP address designated for the client request, the system further comprising a network address translator translating the source IP address designated for the client request to the IP address for the client within the private network.

Other Reference Publication - OREF (3):

Brochure: "WSD-NP A Powerful Global Load **Balancing** Solution", RadWare Ltd., 1997.

US006167438A

United States Patent [19]

Yates et al.

[11] **Patent Number: 6,167,438**

[45] **Date of Patent: Dec. 26, 2000**

[54] **METHOD AND SYSTEM FOR DISTRIBUTED CACHING, PREFETCHING AND REPLICATION**

[75] Inventors: **David J. Yates**, Norwood; **Abdelsalam A. Heddaya**, Waltham; **Sulaiman A. Mirdad**, Quincy, all of Mass.

[73] Assignee: **Trustees of Boston University**, Boston, Mass.

[21] Appl. No.: **08/861,934**

[22] Filed: **May 22, 1997**

[51] **Int. Cl.**^{7} **G06F 12/00**

[52] **U.S. Cl.** **709/216**; 709/223; 709/217; 711/118

[58] **Field of Search** 711/122, 113, 711/118, 130; 709/216, 203, 217, 223

[56] **References Cited**

U.S. PATENT DOCUMENTS

5,852,717	12/1998	Bhide et al.	709/203
5,864,852	1/1999	Luotonen	707/10
5,896,506	4/1999	Ali et al.	709/213
5,924,116	7/1999	Aggarwal et al.	711/122
5,931,912	8/1999	Wu et al.	709/224
5,935,207	8/1999	Logue	709/219
5,940,594	8/1999	Ali et al.	709/203
5,941,988	8/1999	Bhagwat et al.	713/201

FOREIGN PATENT DOCUMENTS

93/24890 12/1993 WIPO .

OTHER PUBLICATIONS

Heddaya, Abdelsalam and Mirdad, Sulaiman "Wave:Wide-Area Virtual Environment for Distributing Published Documents"*ACM SIGCOMM Workshop on Middleware, Cambridge, MA,* Aug. 28–29 (1995).

Heddaya, Abdelsalam and Mirdad, Sulaiman "WebWave: Globally Load Balanced Fully Distributed Caching of Hot Published Documents" *Boston University, Computer Science Department, BU-CS-96-024, Abstract,* (Oct. 15, 1996).

Heddaya, Abdelsalam, et al. "Diffusion-based Caching Along Routing Paths" *Boston University, Computer Science Department Abstract*, (Apr. 30, 1997).

Chankhunthod, Anawat, et al. "A Hierarchical Internet Object Cache" *Proc. USENIX* (1996).

Cormack, Andrew "Web Caching" *Survey from UK National Cache Project*, (Sep., 1996).

Fielding, R., et al. "Hypertext Transfer Protocol—HTTP/1.1" *Standards Track*, 1–94 (Jan., 1997).

"A Distributed Testbed for National Information Provisioning" *NLANR* (1996).

"Netscape Proxy Server" *Netscape Server Central* (1997).

"Squid Internet Object Cache" *features.html by Danial O'Callaghan <danny@hilink.com.au>*.

C. Mic Bowman et al, Harvest: a scalable, customizable discovery and access system Mar. 1995.

Marc Abrams et al, Caching Proxies: Limitations and Potentials Oct. 1995.

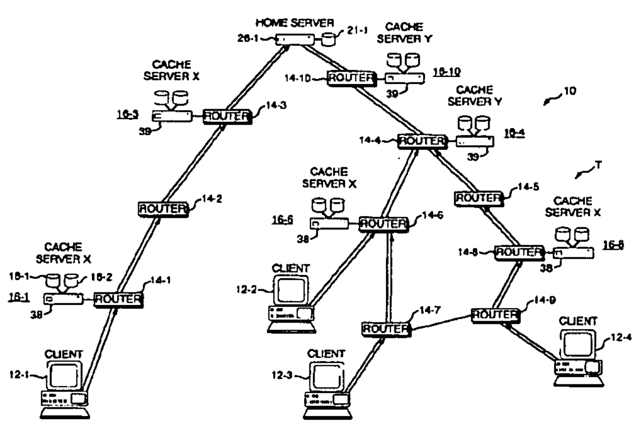
(List continued on next page.)

Primary Examiner—Glenton B. Burgess
Assistant Examiner—Abdullahi E. Salad
Attorney, Agent, or Firm—Hamilton, Brook, Smith & Reynolds, P.C.

[57] **ABSTRACT**

A technique for automatic, transparent, distributed, scalable and robust caching, prefetching, and replication in a computer network that request messages for a particular document follow paths from the clients to a home server that form a routing graph. Client request messages are routed up the graph towards the home server as would normally occur in the absence of caching. However, cache servers are located along the route, and may intercept requests if they can be serviced. In order to be able to service requests in this manner without departing from standard network protocols, the cache server needs to be able to insert a packet filter into the router associated with it, and needs also to proxy for the home server from the perspective of the client. Cache servers may cooperate to service client requests by caching and discarding documents based on its local load, the load on its neighboring caches, attached communication path load, and on document popularity. The cache servers can also implement security schemes and other document transformation features.

75 Claims, 10 Drawing Sheets



OTHER PUBLICATIONS

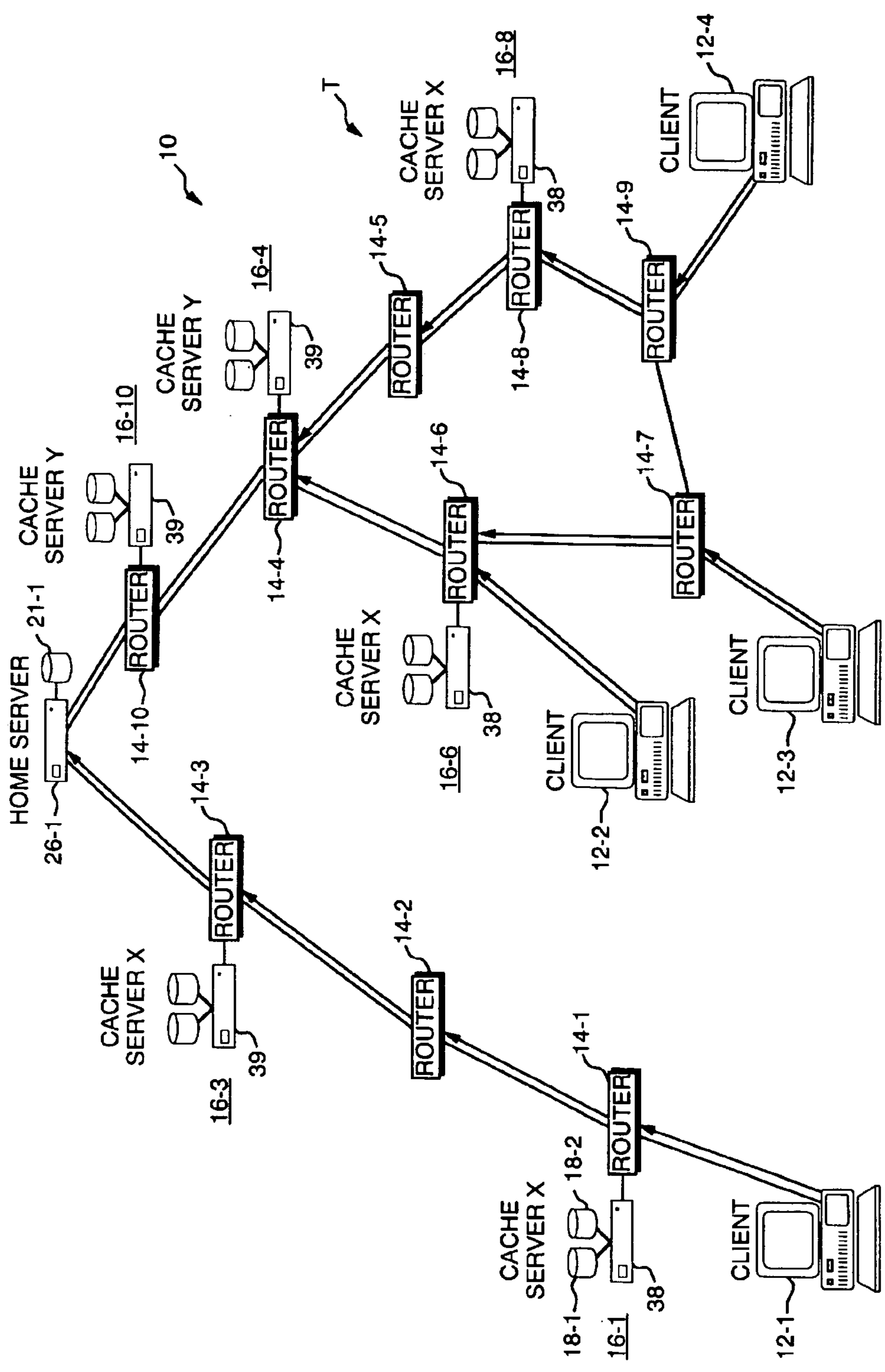
Heddaya, A., et al., "WebWave: Globally Load Balanced fully distributed Caching of Hot Published Documents," Technical Report BU–CS–96–024, Boston University, CS Department, (10/96), downloaded from Internet at: http://www.cs.bu.edu/techreports/abstracts/96–024.

Heddaya, A., et al., "WebWave: Globally Load Balanced Fully Distributed Caching of Hot Published Documents," *Proceedings of the 17th International Conference on Distributed Computing Systems* (Cat. No. 97CB36053), Proceedings of 17th International Conference on Distributed Computing Systems, Baltimore, MD, pp. 160–168 (May 27–30, 1997).

Shrikumar, H., et al., "Thinternet: life at the end of a tether," *Computer Networks and ISDN Systems*, 27 (3):375–385 (Dec. 1994).

Bolot, J., et al., "Performance engineering of the World Wide Web: Application to dimensioning and cache design," *Computer Networks and ISDN Systems*, 28(11):1397–1405 (May 1996).

Braun, H., et al., "Web traffic characterization: an assessment of the impact of caching documents from NCSA's web server," *Computer Networks and ISDN Systems*, 28(1):37–51 (Dec. 1995).



10
T
HOME SERVER
26-1
21-1
ROUTER
14-1
14-2
14-3
14-4
14-5
14-6
14-7
14-8
14-9
14-10
CACHE SERVER X
CACHE SERVER Y
16-1
16-3
16-4
16-6
16-8
16-10
18-1
18-2
38
39
CLIENT
12-1
12-2
12-3
12-4

FIG. 1

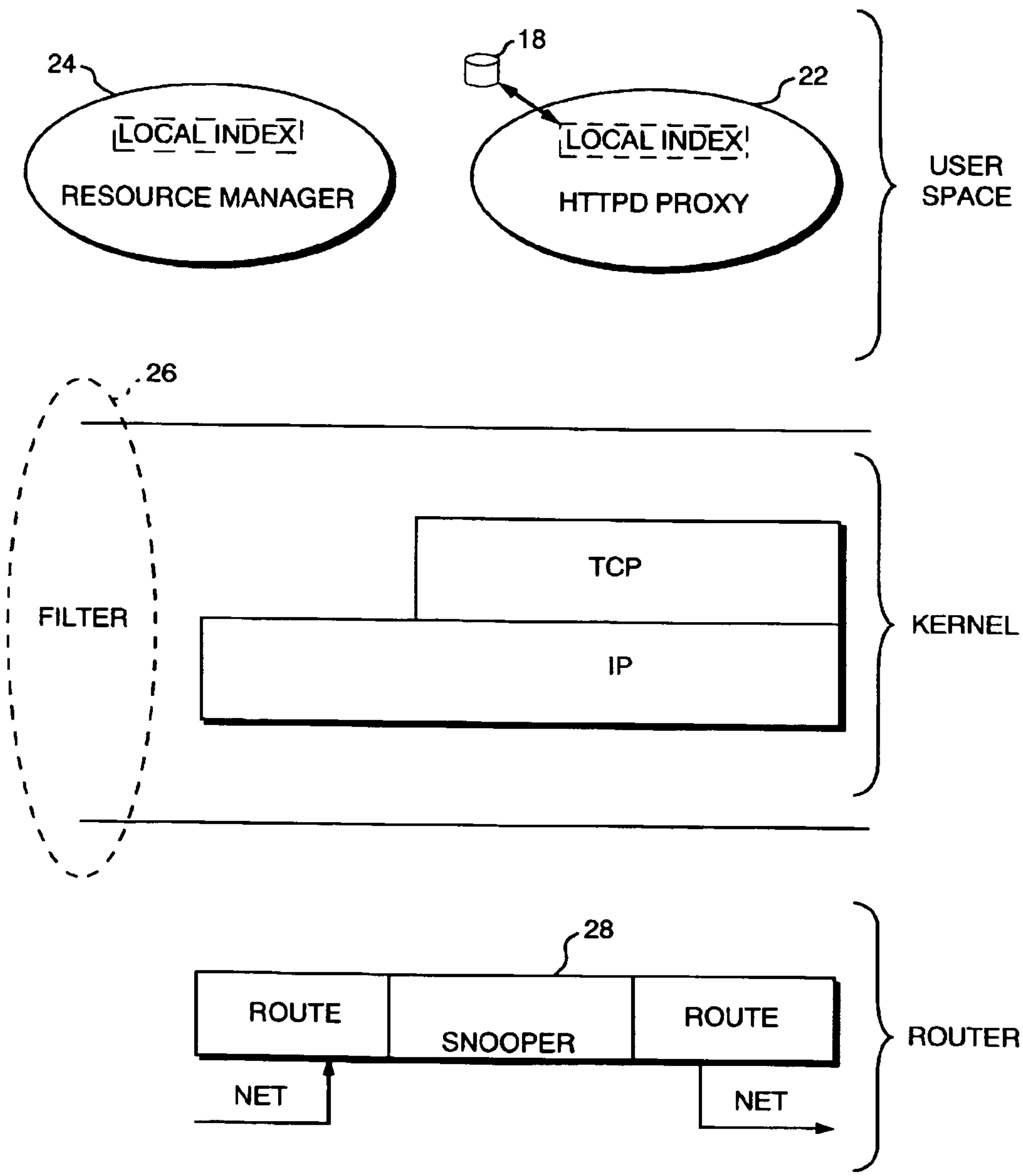

18
24
22
LOCAL INDEX
RESOURCE MANAGER
LOCAL INDEX
HTTPD PROXY
USER SPACE
26
FILTER
TCP
IP
KERNEL
28
ROUTE
SNOOPER
ROUTE
ROUTER
NET
NET

FIG. 2

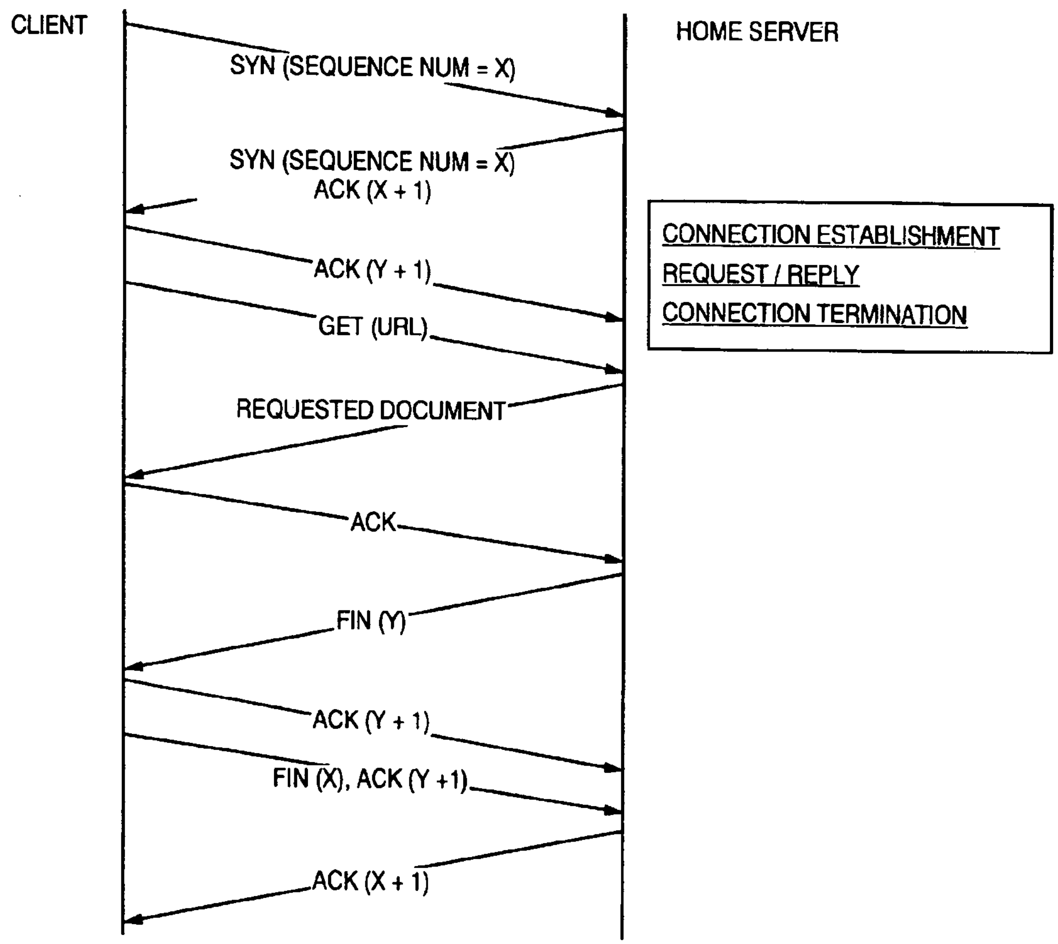

CLIENT
HOME SERVER
SYN (SEQUENCE NUM = X)
SYN (SEQUENCE NUM = X)
ACK (X + 1)
CONNECTION ESTABLISHMENT
REQUEST / REPLY
CONNECTION TERMINATION
ACK (Y + 1)
GET (URL)
REQUESTED DOCUMENT
ACK
FIN (Y)
ACK (Y + 1)
FIN (X), ACK (Y +1)
ACK (X + 1)

FIG.3

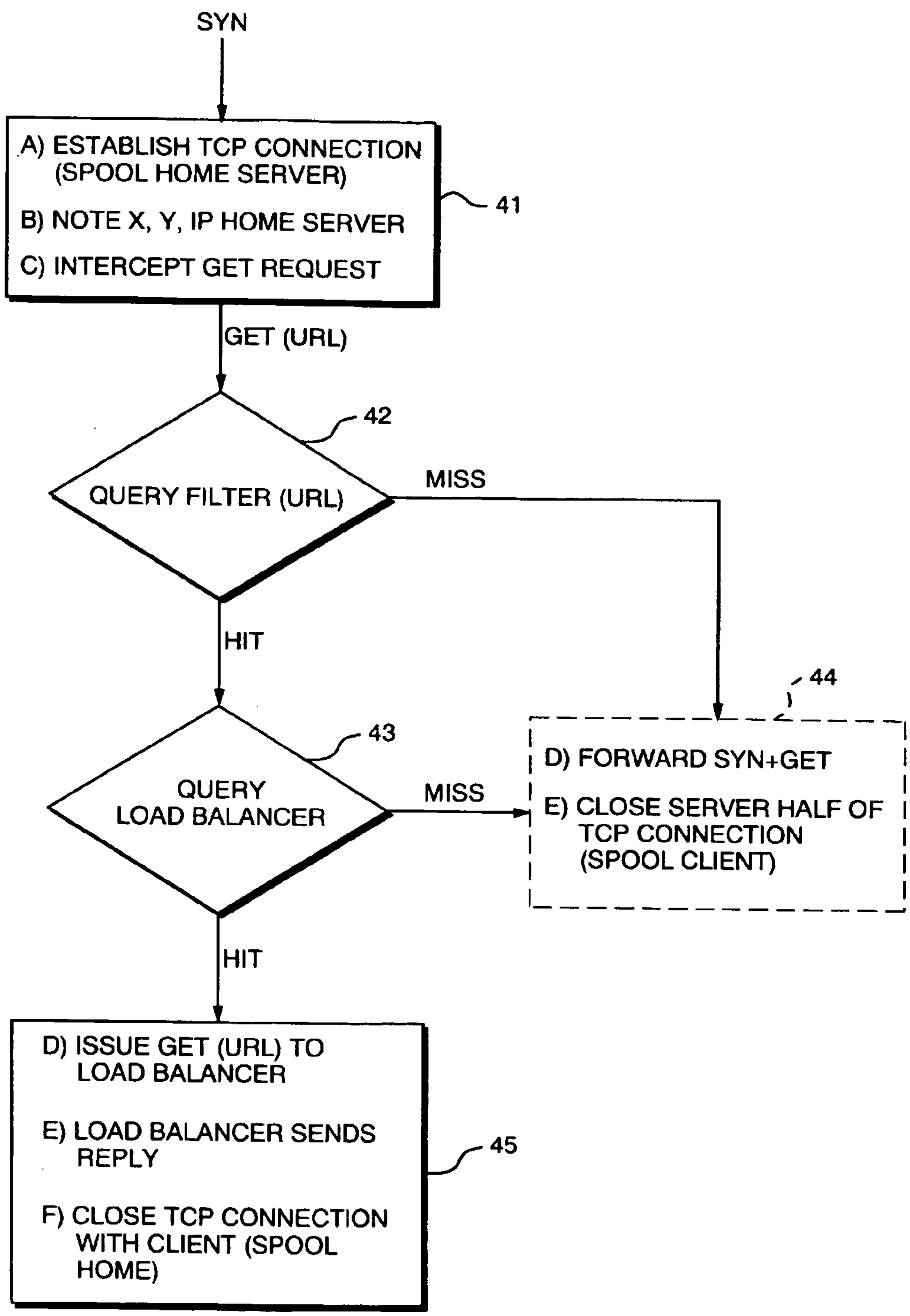

SYN
A) ESTABLISH TCP CONNECTION (SPOOL HOME SERVER)
B) NOTE X, Y, IP HOME SERVER
C) INTERCEPT GET REQUEST
41
GET (URL)
42
QUERY FILTER (URL)
MISS
HIT
44
43
QUERY LOAD BALANCER
MISS
D) FORWARD SYN+GET
E) CLOSE SERVER HALF OF TCP CONNECTION (SPOOL CLIENT)
HIT
D) ISSUE GET (URL) TO LOAD BALANCER
E) LOAD BALANCER SENDS REPLY
F) CLOSE TCP CONNECTION WITH CLIENT (SPOOL HOME)
45

FIG. 4

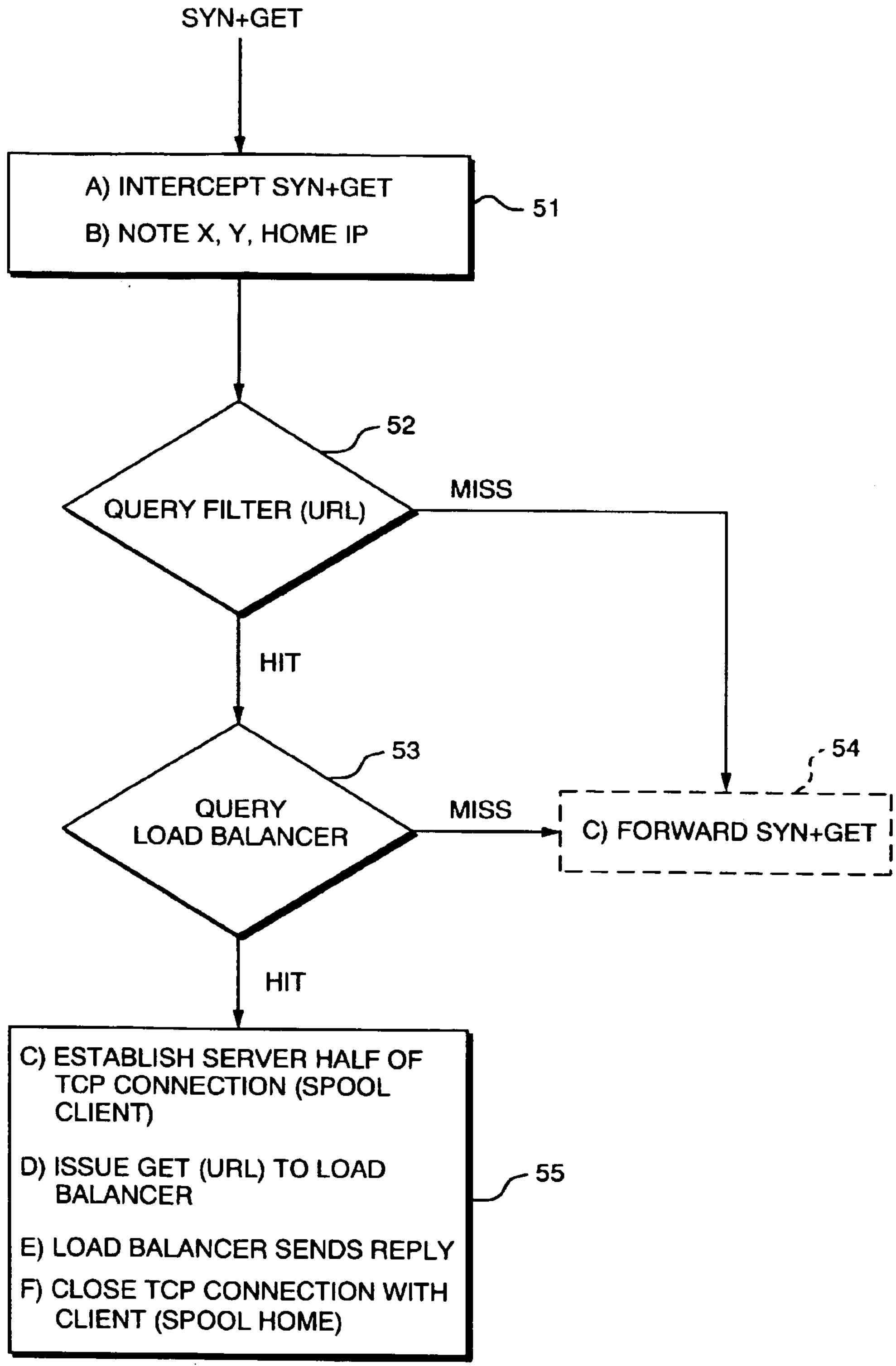

SYN+GET
A) INTERCEPT SYN+GET
B) NOTE X, Y, HOME IP
51
52
QUERY FILTER (URL)
MISS
HIT
53
QUERY
LOAD BALANCER
MISS
54
C) FORWARD SYN+GET
HIT
C) ESTABLISH SERVER HALF OF TCP CONNECTION (SPOOL CLIENT)
D) ISSUE GET (URL) TO LOAD BALANCER
E) LOAD BALANCER SENDS REPLY
F) CLOSE TCP CONNECTION WITH CLIENT (SPOOL HOME)
55

FIG. 5

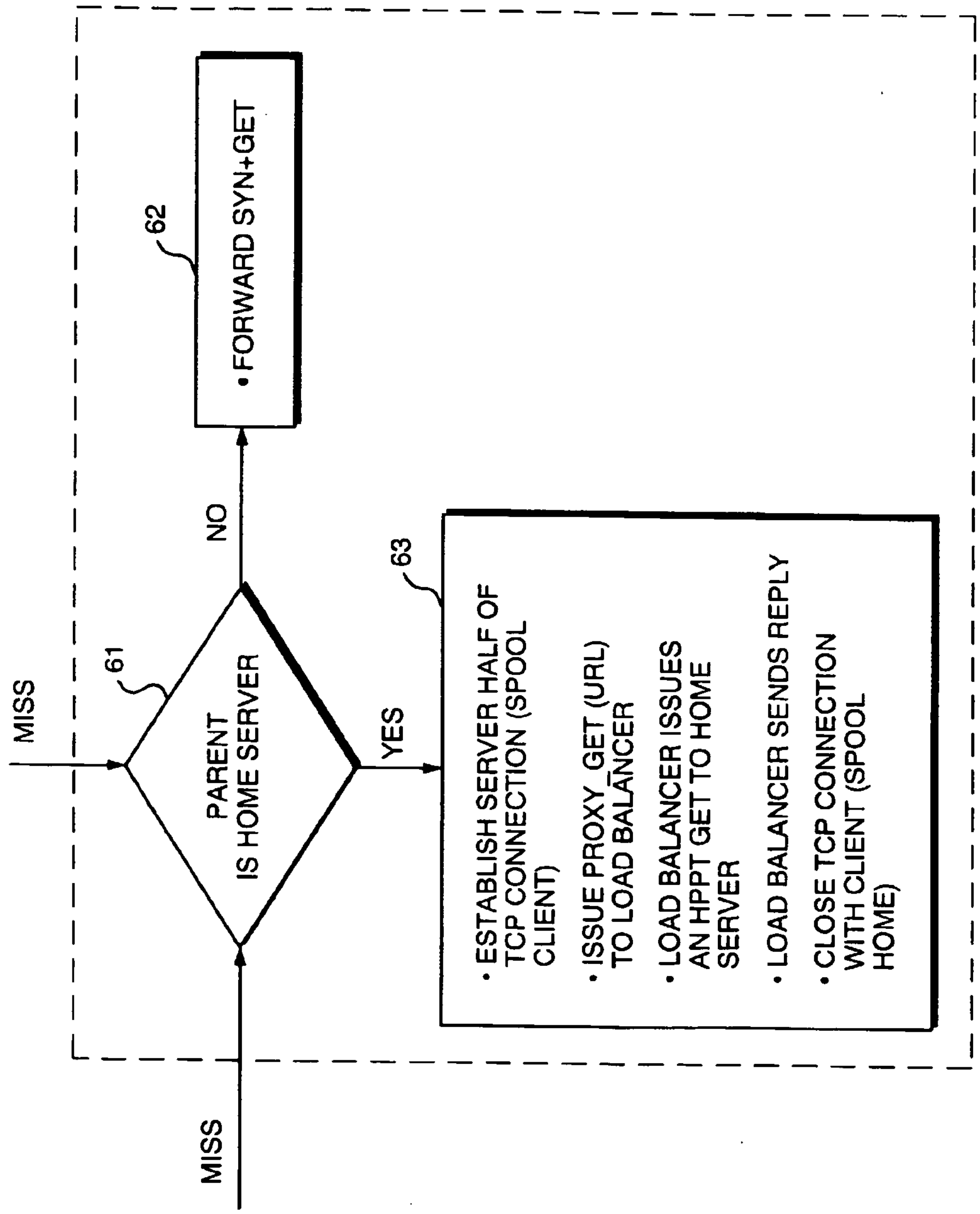

MISS
MISS
61
PARENT IS HOME SERVER
NO
YES
62
• FORWARD SYN+GET
63
• ESTABLISH SERVER HALF OF TCP CONNECTION (SPOOL CLIENT)
• ISSUE PROXY_GET (URL) TO LOAD BALANCER
• LOAD BALANCER ISSUES AN HPPT GET TO HOME SERVER
• LOAD BALANCER SENDS REPLY
• CLOSE TCP CONNECTION WITH CLIENT (SPOOL HOME)

FIG. 6

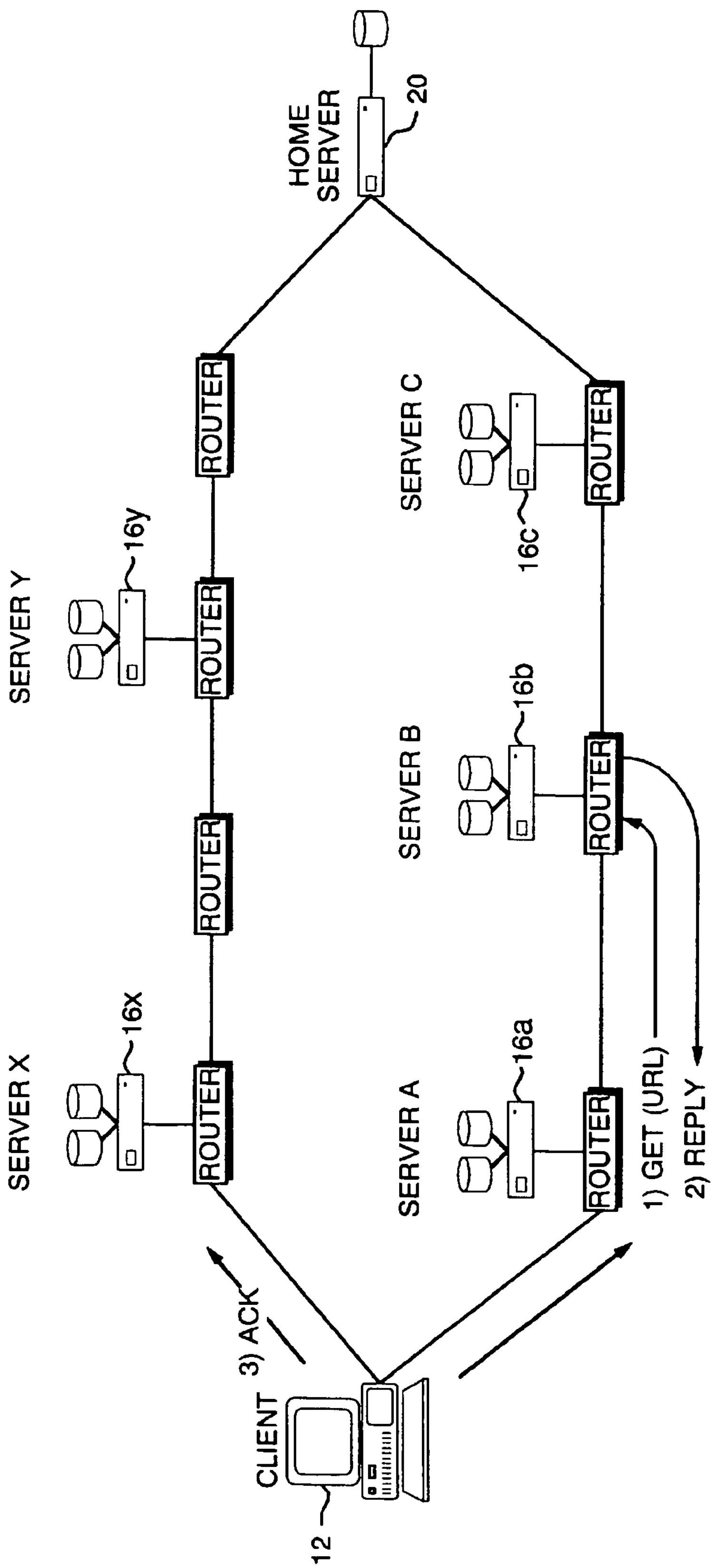

HOME SERVER
20
SERVER X
16X
SERVER Y
16y
SERVER A
16a
SERVER B
16b
SERVER C
16c
ROUTER
ROUTER
ROUTER
ROUTER
ROUTER
ROUTER
ROUTER
1) GET (URL)
2) REPLY
3) ACK
CLIENT
12

FIG. 7

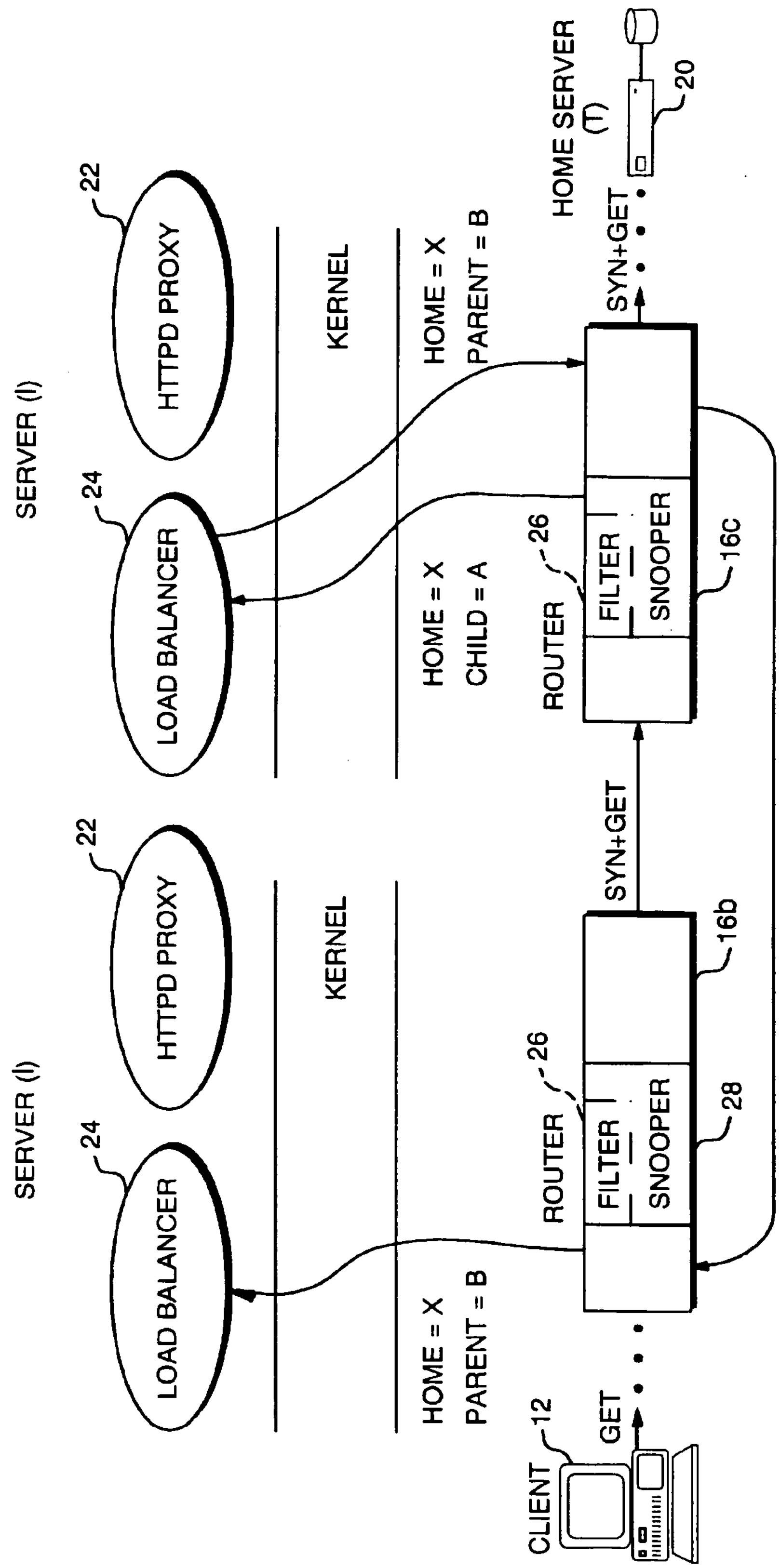

SERVER (I)
LOAD BALANCER
24
HTTPD PROXY
22
KERNEL
HOME = X
PARENT = B
CLIENT
12
GET
ROUTER
FILTER
SNOOPER
26
28
16b
SYN+GET
SERVER (I)
LOAD BALANCER
24
HTTPD PROXY
22
KERNEL
HOME = X
CHILD = A
HOME = X
PARENT = B
ROUTER
FILTER
SNOOPER
26
16C
SYN+GET
HOME SERVER
(T)
20

FIG.8

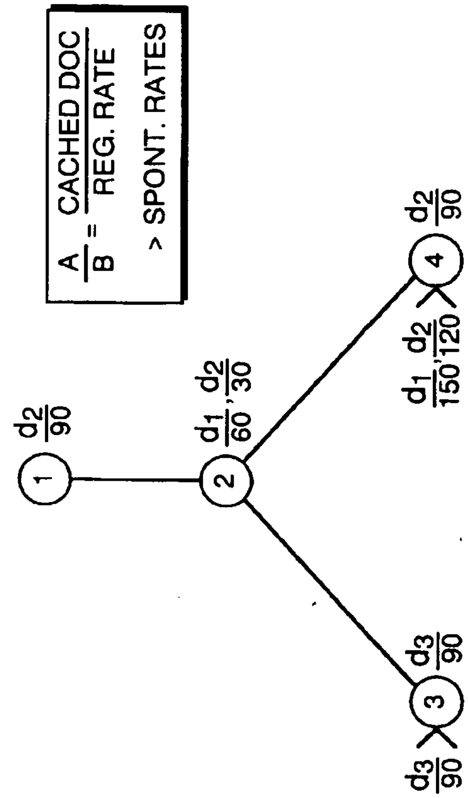


FIG.9B

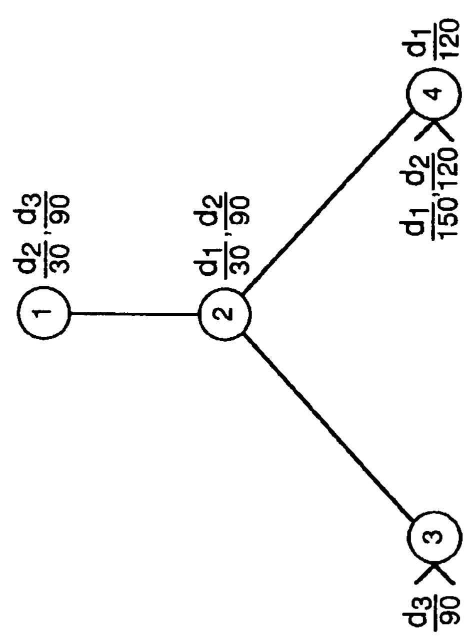


FIG.9A

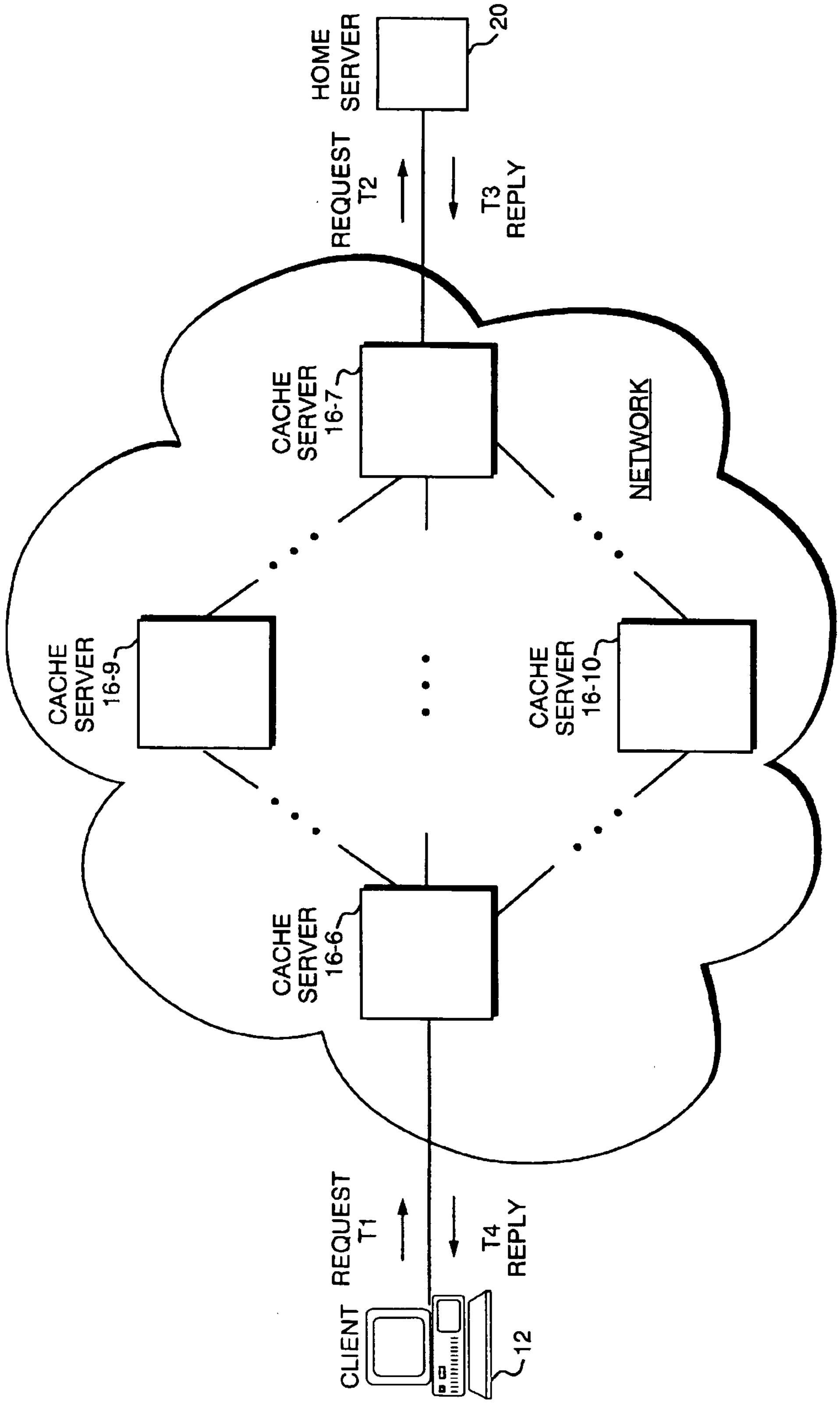

HOME SERVER
20
REQUEST T2
T3 REPLY
CACHE SERVER 16-7
NETWORK
CACHE SERVER 16-9
CACHE SERVER 16-10
CACHE SERVER 16-6
REQUEST T1
T4 REPLY
CLIENT
12

FIG. 10

METHOD AND SYSTEM FOR DISTRIBUTED CACHING, PREFETCHING AND REPLICATION

BACKGROUND

Computer networks, such as the Internet, private intranets, extranets, and virtual private networks, are increasingly being used for a variety of endeavors including the storage and retrieval of information, communication, electronic commerce, entertainment, and other applications. In these networks certain computers, known as servers, are used to store and supply information. One type of server, known as a host or home server, provides access to information such as data or programs stored in various computer file formats but generically referred to herein as a "document". While in the Internet the documents are typically primarily composed of text and graphics, each such document can actually be a highly formatted computer file containing data structures that are a repository for a variety of information including text, tables, graphic images, sounds, motion pictures, animations, computer program code, and/or many other types of digitized information.

Other computers in the network, known as clients, allow a user to access a document by requesting that a copy be sent by the home server over the network to the client. In order for a client to obtain information from a home server, each document typically has an address by which it can be referenced. For example, in the context of the Internet and within the communication protocol known as Hyper Text Transfer Protocol (HTTP), the address is typically an alphanumeric string, known as a Uniform Resource Locator (URL), that specifies (a) an address of the home server from which to obtain the information in the form of a name or a numerical address, and (b) a local information text string that identifies the information requested by the client, which may be a file name, a search request, or other identification.

After the user specifies a URL to the client computer, the address portion of the URL is sent over the network to a naming service such as the Domain Name Service (DNS) in order to obtain instructions for how to establish a connection with the correct home server. Once the connection with the server is established, the client can then retrieve the desired document by passing the local information text string over the network directly to the home server. The server then retrieves the document from its local disk or memory storage and transmits the document over the network to the client. The network connection between the home server and the client is then terminated.

Computer and network industry analysts and experts are presently quite concerned that traffic on the Internet is becoming so heavy that the very nature of the way in which it is possible to use the Internet may change. In particular, many individuals now believe that the Internet is intolerably slow and is no longer a reliable entity for the exchange of information in a timely fashion.

The present bottlenecks are no doubt the result of exponential increases in the number of users as well as in the number of complex documents such as multimedia files being sent. It might appear that the answer is simply to add more bandwidth to the physical connections between servers and clients. This will come, however, only at the expense of installing high bandwidth interconnection hardware, such as coaxial or fiber optic cable and associated modems and the like, into homes and neighborhoods around the world.

Furthermore, added bandwidth by itself perhaps would not guarantee that performance would improve. In particular, large multimedia files such as for video entertainment would still potentially displace higher priority types of data, such as corporate E-mails. Unfortunately, bandwidth allocation schemes are difficult to implement, short of modifying existing network communication protocols. The communication technology used on the Internet, called TCP/IP, is a simple, elegant protocol that allows people running many different types of computers such as Apple Macintoshes, IBM-compatible PCs, and UNIX workstations to share data. While there are ambitious proposals to extend the TCP/IP protocol so that the address can include information about packet content, these proposals are technologically complex and would require coordination between operators of many thousands of computer networks. To expect that modifications will be made to existing TCP/IP protocols is thus perhaps unrealistic.

An approach taken by some has been to recognize that the rapidly growing use of the Internet will continue to outstrip server capacity as well as the bandwidth capacity of the communication media. These schemes begin with the premise that the basic client-server model (where clients connect directly to home servers) is wasteful of resources, especially for information which needs to be distributed widely from a single home server to many clients. There are indeed, many examples of where Internet servers have simply failed because of their inability to cope with the unexpected demand placed upon them.

To alleviate the demand on home servers, large central document caches may be used. Caches are an attempt to reduce the waste of repeated requests for the same document from many clients to a particular home server. By intercepting parallel requests, a cache can be used to serve copies of the same document to multiple client locations.

From the client's point of view, the interaction with a cache typically occurs in a manner which is transparent to the user, but which is slightly different from a network messaging standpoint. The difference is that when the address portion of the request is submitted to the Domain Name Service (DNS) to look up the information needed to connect to the home server, the DNS has been programmed to return the address of a cache instead of the actual original home server.

Alternatively, a server node, acting as a proxy for the client, may issue probe messages to search for a cache copy. Once a cache copy is found at a particular node in the network, the request is then forwarded to that node. For example, under the auspices of the National Science Foundation, document caches have been placed at various locations in the United States in order to eliminate bottlenecks at cross-oceanic network connections. Generally, certain of these caches located on the West Coast handle requests for documents from the Asia-Pacific and South American countries, and a number of those located on the East Coast handle requests for documents from Europe. Other of these national caches handle requests for popular documents located throughout the United States.

However, such caching techniques do not necessarily or even typically achieve optimum distribution of document request loading. In particular, in order for the caches to be most effective, the DNS name service or other message routing mechanism must be appropriately modified to intercept requests for documents for which the expected popularity is high. The introduction of cache copies thus increases the communication overhead of name resolution, because of the need to locate the transient copies. The name service must register these copies as they come into

existence, disseminate this information to distribute demand for the documents, and ensure the timely removal of records for deleted cache copies. Often times, the cache lookup order is fixed, and/or changes in document distribution must be implemented by human intervention.

Unfortunately, frequent and pronounced changes in request patterns can force the identity, location, and even the number, of cache copies to be highly transient. The resulting need for updating of cache directories means that they cannot typically be replicated efficiently on a large scale, which can thus turn the name service itself into a bottleneck.

Another possible approach to implementing caches is to change the client/server interaction protocol so that clients proactively identify suitable cache copies using a fully distributed protocol, for example, by issuing probes in randomized directions. Aside from the complexity of modifying existing protocols and message cost introduced by such an approach, such a scheme also adds one or more round trip delays to the total document service latency perceived by the client.

SUMMARY OF THE INVENTION

The present invention is an automatic, distributed, and transparent caching scheme that exploits the fact that the paths that document requests follow through a computer network from a client to a particular document on a particular home server naturally form a routing graph, or tree.

According to the invention, cache servers are placed throughout the network, such that if a document request can be fulfilled at some intermediate node along the routing graph, it will be serviced by the intermediate node returning the cached document to the client. The document request messages are thus responded to before they ever reach the home server. Since document request messages are permitted to be routed from clients in the direction of the home server up the routing graph in the same manner as would occur in the absence of caching, naming services do not need modification.

In order to be able to service requests in this manner without departing from standard network protocols, a cache server includes a packet filter in its associated router. The filter extracts document request packets that are highly likely to hit in the associated cache.

The cache server also preferably acts as a communication protocol proxy for the home server. That is, as part of fulfilling document request messages at the intermediate node locations, the client is sent appropriate messages, depending upon the communication protocol in use, to spoof the client into believing that the document was actually received from the home server.

The invention also provides a manner in which caching servers may cooperate to service client requests. In particular, each server has the ability to cache and discard documents based on its local load, the load on its neighboring caches, adjacent communication path load, and on document popularity. For example, each server maintains an estimate of the load at its neighbors, and communicates its own load estimate to neighboring cache servers. If a cache server notices that it is overloaded with respect to any of its neighbors, it offloads or transfers a fraction of its work to its under loaded neighbors. To do so, a cache server also preferably learns the identity of its neighboring upstream (or parent)and downstream (or child) nodes on the routing graph that is rooted at a given home server.

There are several advantages to the basic concepts of a document caching system according to the invention.

First, the approach does not need to request an address lookup from a cache directory, to redirect document requests, or to otherwise probe other elements of the network to locate cache copies. Location of the cache copy thus occurs fortuitously, along the natural path that the request message follows anyway. The client thus does not experience delays or bottlenecks associated with waiting for other entities in the network to find appropriate cache copies.

In addition, the system as a whole permits cache copies of documents to diffuse through the network as needed, which in turn diffuses bottlenecks at the caches and well as along the communication paths.

There is also a corresponding reduction in network bandwidth consumption and response time, because cache copies are always placed nearer to the original server than to the client, Document request messages and the documents themselves therefore typically do not need to travel the full distance between the server and each client every time they are requested. Hence, overall network bandwidth is conserved, response times are reduced, and load is more globally balanced.

The invention thus not only helps to dampen differences in the load demand on the host servers, but also reduces the load on network communication resources, without requiring any modification to existing network protocols.

Furthermore, because cache copies are distributed through the network, there is no single expected point of failure of the caching system, and the system is robust and fail-safe.

The technique is also scalable, in the sense that as more cache servers are added, both clients and servers experience a likewise benefit.

The invention can be used to implement additional functionality in a network. These various functions are a direct result of the fact that the packet filter implemented at the router associated with the cache servers can also do more than simply divert requests for copies of documents to the local cache.

For example, popularity statistics are of necessity collected by the cache servers, in order to control the choice of what documents to cache. Each cache server thus keeps track of how many references to a particular document are received and from where they are received and can determine aggregate request amounts and request rates. This data can be collected at a central location, such as by network region, so it is then possible for a publisher of documents to not only obtain statistics about the hit rate on their material but also where in the network the hits are coming from. This is important not only for document popularity analysis, but also electronic commerce and intellectual property tracking.

The cache servers can also be used to host replicas of popular documents such as databases, search engine index files, and the like, by acting as load splitters from the service provider perspective. In other words, database providers can arrange to have their documents placed into the network, pushing out data closer to the clients that desire access to it, wherever the best placements might be.

A set of security features may also be readily attached to the cache servers.

One such feature is the authentication of the sources of request messages and other information. This is possible because the cache servers maintain information as to the physical source of document request messages and of the documents themselves. The mechanism also arises from the fact that the nodes have a filter and a packet router. The filter

and packet router may be used not only to keep track of how to redirect requests to cache copies, but also to restrict access to the cache copies, such as by authenticating the request for the information. The invention also enables various types of document distribution in a the network. For example, the invention permits document compression, which is another form of conserving bandwidth, or encryption, as long as a particular server and client node are committed to communicating by using cache servers, e.g., the first and last nodes along the path between the client and the server in the network contain cache servers.

The invention also permits the efficient caching of dynamic content documents. Such dynamic content documents are of the type where what is to be returned to the client changes on the fly, typically in accordance with program instructions. When the invention recognizes the existence of dynamic content documents in its cache, it caches not only the data for the document, but also allows for fetching and executing of programs that specify how the document is to be displayed or when the data is retrieved.

The invention also improves the delivery of stored continuous media such as audio and video data files since the number of network nodes between a server and a client are reduced.

BRIEF DESCRIPTION OF THE DRAWINGS

For a more complete understanding of the advantages provided by the invention, reference should be had to the following detailed description together with the accompanying drawings, in which:

FIG. 1 depicts a typical computer network showing a request path for a single document and the location of cache servers along the path according to the invention;

FIG. 2 is a communication layer diagram illustrating how a resource manager, protocol proxy, and snooper are used to implement the invention;

FIG. 3 shows the typical stages in a document request over the network;

FIG. 4 is a flow chart of the operations performed by a leaf server located on the routing path according to the invention;

FIG. 5 is a flow chart of the operations performed by a intermediate non-leaf cache server;

FIG. 6 is a flow chart of the operations performed by a last cache server on the routing path;

FIG. 7 illustrates the interception of a document request message by an intermediate server;

FIG. 8 also illustrates the interception of a document request message in more detail;

FIGS. 9(*a*) and 9(*b*) illustrate how diffusion can proceed in a worst case client request scenario; and

FIG. 10 illustrates how the cache servers may implement document transformation functions.

DETAILED DESCRIPTION OF PREFERRED EMBODIMENTS

1. Introduction

Turning attention now to FIG. 1, a computer network 10 such as the Internet, extranet, private intranet, virtual private network, local area network, or any other type of computer network consists of a number of network entities including client computers 12-1, 12-2, 12-3, 12-4 (collectively, clients 12), routers 14-1, 14-2, . . . , 14-10, cache servers 16-1, 16-3, 16-4, 16-6, 16-8, and 16-10, and home server 20. The network may make use of any and various types of physical layer signal transmission media such as public and private telephone wires, microwave links, cellular and wireless, satellite links, and other types of data transmission.

In the illustrated network, certain routers 14 have associated with them cache servers 16, whereas other routers do not have associated cache servers. The cache servers 16 include various types of storage for documents in the form of a cache storage 18-1 which may include disk storage 18-1-1 and/or memory storage 18-1-2.

The clients 12 and home server 20 operate as in the prior art to permit distribution of a wide variety of information, typically in the form of "documents". Such documents may actually contain text, graphics, pictures, audio, video, computer programs and any number of types of information that can be stored in a computer file or parts of a computer file. Furthermore, certain documents may be produced at the time that access is requested to them, by executing a program.

It will be assumed in the following discussion that the network 10 is the Internet, that the information is encoded in the form of the Hyper Text Transfer Protocol (HTTP) documents, and that document request messages are sent in the form of Uniform Resource Locators (URLs) using the TCP/IP layered protocol. This is with the understanding that other types of wired, switched, and wireless networks, and other types of protocols such as FTP, Gopher, SMTP, NNTP, etc. may make advantageous use of the invention. In addition, although the invention is discussed in the context of a client-server type of communication model, it should be understood that the principals of the invention are equally applicable to peer-to-peer networks.

A request message for a particular document, for example, originates at one of the client computers, such as client 12-1. The message is a request by the client 12 for the home server 20 to send a copy of document that is presently stored at the home server 20 location such as on a disk. The document request message is passed through one or more routers 14, such as routers 14-1, 14-2, 14-3, in the direction of the illustrated arrows, on its way to the home server 20.

In networks such as the Internet, document request messages may pass through as many as fifteen or more nodes or "hops" through routers 14 before reaching their intended destination. Requests for the same document from other clients, such as clients 12-2, 12-3, or 12-4 also pass through different routers 14 on their way to the home server 20 at the same time.

It should also be understood that although the routers 14 and cache servers 16 are shown as separate elements in FIG. 1, that their functionality may be combined into a single element.

A model is useful for understanding the nature of how requests from multiple clients for one particular document travel across a path the computer network 10. The model is that structure, T, which is induced by the effect of routing algorithm on the document request messages as they travel through the network to the home server 20. As shown in FIG. 1, the home server 20 can thus be thought of as being at the root node of the structure, T, with document requests originating at the leaf node levels farthest away from the root, namely at clients 12-1, 12-2, . . . , 12-4. The structure T also includes many intermediate nodes which are located the routers 14.

While the structure T of the set of paths that client requests follow towards a given home server 20 is accurately and generally described as a data directed, acyclic graph, the present exposition does not benefit from the added complexity. In particular, when a single particular document is considered as being located at only one home server, the

structure can be referred to as a tree with a single root. With that understanding we use the term tree to describe the structure T herein, with the understanding that a graph model may also be used. With this model in mind, the entire Internet can be thought of as a forest of trees or graphs, each rooted at a different home server 20 which is responsible for providing an authoritative permanent copy of some set of documents.

In accordance with the invention, copies of documents are located in the network at cache servers 16. According to the invention, the placement of cache copies, and hence the diffusion of load, is constrained to nodes in the tree structure, T. This avoids the need for clients to lookup the locations of cache copies, either by directly contacting the home server 20, or a naming service such as a Domain Name Service (DNS), or by probing the network in search of apropriate cache copies.

The present invention also assumes that cache servers 16 lie on the path along the tree that document request messages would naturally take from the client 12 to the home server 20, with the cache servers 16 cooperating to off-load excess load at the home server 20, or to diffuse other potential performance bottlenecks such as communication links themselves. In effect, the routers 14 having associated cache servers 16 inspect document request message packets as they fly-by and intercept any request for which it may be possible to fulfill by providing a cached document instead.

In a most general description of the operation of the invention, document request messages travel up the tree T, from a client at which it originated, such as client 12-3, towards the home server 20. Certain routers encountered by the document request message along the way, such as router 14-7, do not have local cache servers 16, and thus simply pass the document request message up to Findeathe next router in the tree, such as router 14-6.

However, certain other routers, such as router 14-6, do have a local cache server 16-6, in which case the document request message is examined to determine if it is seeking a document located in the local cache store 18. If a cache copy is encountered at cache server 16-6, then that copy is returned to the client 12, and the request message is not permitted to continue on its way to the home server 20. If however, a cache copy is not encountered at the particular cache server 16-6, the request message continues to the next router 14-4 on the path to the home server 20.

When a request message packet enters a router 14, the router first passes the request message to a portion of its software referred to herein as the filter code. The filter code in the router 14 is updated as necessary by the local cache server 16. The filter code depends on the types of packets, the cache contents, the load at the local cache server 16, or the load on the attached communication links. The filter causes the interception of the packet (for an attempted service by the local cache server 16) or passes the packet back to the router 14 to determine the next hop the packet should take on its way to the home server 20.

Ideally, the implementation of the cache servers 16 is such that no changes are required to the normal operating mode of either clients 12 or servers 20. Another goal is to have a design that can be gradually deployed into the existing infrastructure of the network 10. This also requires that any new mechanisms preferably be compatible with existing communication protocols.

To accomplish this a cache server 16 and associated router 14 preferably consist of four functional building blocks, as shown in the layer diagram of FIG. 2. At a relatively higher layer protocol level, such as the application layer, the cache server 16 includes an HTTP proxy 22 and a resource manager 24. At a lower layer, such as the physical layer, the router typically implements a packet filter 26 and an IP proxy or snooper 28.

The HTTP proxy 22 implements a standard HTTP protocol with responsibilities including storage management and the maintenance of the index structures necessary for accessing cached documents. If the HTTP proxy 22 receives a request for a document not located in the local cache 18, it requests the document from the home server 20 and respond to the request when the document arrives. The HTTP proxy 22 is configured to cache documents only as instructed by the resource manager 24.

While FIG. 2 shows two types of proxying, namely at the HTTP and IP level, it should be understood that the implementation can also include proxying at other layers, including the application layer, IP layer, or some other layer in between, such as a transport, session, presentation, or other layer.

The resource manager 24 implements a protocol to diffuse document copies through the network 10, as will be described in greater detail below. The resource manager 24 is responsible for maintaining state information used by the document load diffusion mechanism. The resource manager may be programmed to not only manage the load on the cache servers 16 themselves, but may also be programmed to manage the traffic on the communication paths used interconnect the routers 14.

To accomplish this load management, or load balancing, the resource manager 24 maintains information about the identity and the load of its neighboring cache servers 30. The details of how neighboring cache server information is maintained is discussed below in Section 3.

In addition, for each document in the cache 18, the resource manager 24 distinguishes between requests received through each of its neighboring cache servers 30. This is done by maintaining a separate hit count for requests received from each neighboring cache server 30. Using such information, the resource manager 24 computes the fraction of excess load to be diffused. Once these fractions are determined, the resource manager 24 informs its under-loaded neighbors 30 which document to cache and the fraction of requests they should undertake. These fractions are also used to generate new filter code to be injected into the associated router 14. A similar process is performed by the under-loaded neighbors 30. If necessary, the resource manager 24 at the under-loaded neighbor 20 informs the attached HTTP proxy 22 to add new documents to its cache 18.

Other responsibilities of the resource manager 24 include neighborhood discovery, propagating load information to the neighboring servers 30, and discovering and recovering from potential barriers to load balancing. These mechanisms are discussed in more detail below.

The routers 14 take an active role in assisting cache servers 16 to achieve cache server and/or communication path balancing goals. This is accomplished by allowing the resource manager 24 to inject functionality into the router 14 in the form of the code that implements the filter 26 and snooper 28. In particular, all packets passing through a router 14 not addressed directly to a host server 20 are first passed to the snooper 28. The snooper 28 inspects a packet and determines its type, destination, and the document requested. Depending on the state of the cache server 16 and packet type, the snooper 28 could intercept the packet or simply forward the packet to the next hop, or router 14, along the intended destination path to the home server 20.

To determine if a requested document is located at the local cache server **16**, the snooper **28** queries the filter **26**. If the filter **26** indicates that the requested document is cached and can be serviced locally, then the packet is intercepted and passed to the resource manager **24**. Otherwise, the packet is passed on to the next hop towards the destination home server **20**.

The snooper **28** is typically aware of the TCP protocol and the structure of both TCP and HTTP packets. Another functionality of the snooper **28** is to extract copies of HTTP request packets and pass them up to the resource manager **24**. This feature is used to assist the resource manager **24** in discovering its neighborhood and recovering from potential barriers.

2. Handling an HTTP Document Request in a TCP/IP Network Current implementations of networks **10** that use HTTP rely on the layered TCP/IP protocol for reliable end-to-end communication between clients **12** and servers **20**. This layering divides the normal processing of a request message into three steps; connection establishment (i.e., TCP-level three way handshake in the form of {SYN} messages), HTTP document request/reply in the form of {GET} messages, and connection termination in the form of {FIN} messages.

This process is depicted in FIG. **3**, where the client **12** first issues a {SYN} message with a sequence number to the home server **20**, and the home server **20** returns a {SYN} message with an acknowledgment {ACK}. In response to this, the client **12** then sends a document request in the form of a {GET} message that includes the URL of the desired document. The document is then forwarded by the home server **20** to the client **12**. After the client **12** returns an acknowledgment, the server **20** and client **12** terminate the connection by exchanging {FIN} and {ACK} messages.

The main hurdle in actually implementing the cache servers **16** as explained above in such an environment is the requirement that they need to identify the document requested by a client **12**. However, as seen in FIG. **3** the URL information is typically advertised by an HTTP client **12** only after a TCP/IP connection has already been established with the home server **20**. One possible solution would thus be to have all such connections be established with the home server **20** and have snoopers **28** at intermediate routers **14** intercept all {GET} packets. Even though this approach might relieve a significant amount of load from a home server, it still required that TCP connections associated with such documents reach the home server **20**, which defeats the purpose of attempting to off-load the home server **20**. During high demand periods, such requests would amount to a flood of {SYN} requests on the home server **20**. In addition, if the initial {SYN} is not intercepted, both establishing and tear down of connections becomes significantly more complicated.

To overcome this hurdle, in the preferred embodiment, intermediate routers **14** have some awareness of the TCP protocol. TCP aware routers **14** are able to detect TCP connection requests to all HTTP servers (i.e., a {SYN} packet directed to the HTTP port), and have the ability to act as a proxy for, or "spoof" the home server **20**.

This functionality is implemented by the snooper **28**. In particular, snoopers **28** located in routers **14** on the path to a home server **20** inspect packets that fly-by, identify such packets, and intercept any {SYN} packets directed to HTTP home servers **20**. As {SYN} packets do not contain any information identifying which document the client **12** intends to request, the snooper **28** acts as a proxy for, or "spoofs" the home server **20**, by establishing a connection between the client **12** and the local transport layer in the cache server **16**, and noting the initial sequence numbers used by both the client **12** and the local transport layer.

After the connection is established the snooper **28** inspects all packets that fly-by, and waits for the corresponding {GET} request. Once the {GET} request arrives the snooper **28** queries the local filter **26** and the resource manager **24** to determine if the requested document is cached. If the document is cached the snooper **28** forwards the HTTP {GET} message to the local resource manager **24**, waits for the resource manager **24** to service the request, and then terminates the connection. Otherwise, the requested document is not cached (i.e., the filter **26** or resource manager **24** missed). Several different approaches may be taken to servicing the document request at this point.

In a first approach, the TCP connection is handed off, wherein the snooper **28** closes the server half of the spoofed TCP connection with the client **12**, and forwards the document request in the form of a composite "piggy back" {SYN+GET} message in the direction of the home server **20**. In addition, the {SYN+GET} message contains all the state information needed to hand-off the server half of the TCP connection to any other intermediate cache server on the path to the home server **20** which happens to cache the requested document.

In a second alternative approach, the snooper may act as a TCP relay, maintaining the TCP connection with the client, and relaying the {SYN+GET} message on a separate connection to the next intermediate cache server on the path to the home server **20**.

The above hand-off process is illustrated in the flow chart of FIG. **4**. This process is carried out by a particular class of cache servers **16** referred to as leaf node servers **38**, which are the cache servers **16** that are on the extreme lower level nodes of the tree T, i.e., the first servers to intercept a {SYN} packet from a client **12**. The leaf node servers **28** in the tree T depicted in FIG. **1** are cache servers **16-1**, **16-6**, and **16-8**.

As shown in step **41** of FIG. **4**, when a leaf node server **38** receives a {SYN} packet, the home server **20** is proxied for, or "spoofed", by establishing a TCP connection directly between the leaf node server **38** and the client **12**. The leaf node server **38** then waits to intercept the corresponding {GET} request from the client **12**.

Note that spoofing thus occurs in the sense that packets exchanged between the client **12** and a cache server **16** are modified by the snooper **28** in the above scenario. In particular, the network address of a cache server **16** which is servicing a request is replaced with the network address of the home server **20** and in a connection hand-off, the sequence numbers of bytes issued by the cache server **16** have to follow the sequence number as determined by the leaf server **38**.

Returning to step **41**, if the requested document passes the cache query test by the filter **28**, and in step **42**, and if the resource manager **22** detects that the document is present in the local cache and will permit access to it, then the document request is serviced locally, in step **45**. In step **45**, the {GET} command is forwarded to the resource manager, which then replies with the requested document. Finally, the TCP connection between the leaf server **38** and the client **12** is closed, by spoofing the home server **20** once again and issuing the closing {FIN} and {ACK} messages to the client.

Otherwise, if there is a miss in step **42** or **43**, the snooper **28** forwards a {SYN+GET} packet in the direction of the home server **20**, and then closes the server half of the spoofed TCP connection, so that another cache server on the

tree may service it if possible. The steps d) and e) in FIG. 4 may be asynchronous events and may typically occur in parallel. The snooper **28** at a leaf server **38** then has to acknowledge the reception of the {GET} request.

In the scenario depicted in FIG. 1, the upstream intermediate non-leaf nodes **39** include those with cache servers **16-3**, **16-4**, and **16-10**. The cache servers **16** located at the non-leaf nodes **39** need to process {SYN+GET} packets in a slightly different manner. In particular, the snooper **28** in a non-leaf node **39** intercepts {SYN+GET} packets only if the requested document is cached and the local cache server **16** has sufficient capacity to service it.

FIG. **5** is a detailed flow chart of this process as performed at the non-leaf intermediate nodes **39**. As shown in step **51**, to service such a request, the snooper **28** first spoofs upon receipt of the {SYN} from the leaf node **38**, and intercepts the following {GET} request. In the following steps **52** and **53**, queries are made to the filter **26** and resource manager **24** as before, to determine if the {GET} can be processed locally.

If the request can be processed locally, step **55** completes the proxying for the home server **20** by establishing the server half of the TCP connection with the client **12**, issuing the {GET} to the resource manager **24**, returning the document to the client **12**, and closing the TCP connection.

If the {GET} message cannot be processed locally, step **54** is executed, where the {SYN+GET} is forwarded to the next node in the tree T.

The main advantage of processing {SYN+GET} packets differently in the intermediate non-leaf nodes **39** is that a TCP connection is only handed-off once to the particular intermediate node **39** that actually has the requested document. Another advantage is that the {SYN+GET} contains all the state information needed for connection hand-off (i.e., no additional state information is exchanged between the snooper **28** at the leaf node server **38** and that at the intermediate node **39** which is actually caching the requested document.)

One drawback of piggy-backing {SYN+GET} packets in this manner is that home servers **20** will not interpret such packets properly without adapting their transport protocol to deal with such packets. To avoid this problem and ensure inter-operability with current network protocols, an additional precaution can be taken by requiring that the snooper **28** located at the last intermediate node **39** before a home server **20** intercept all {SYN+GET} packets. Thus, when none of the leaf node servers **38** or intermediate node servers **39** cache the requested document, the last intermediate server **39** intercepts the {SYN+GET} and relays an explicit HTTP {GET} request to the home server **20**.

To accommodate this case, step **54** of FIG. **5** can be replaced with the processes illustrated in FIG. **6**. In this case, in step **61**, where the next upstream node along the path, T, (or parent node) is not the home server **20**, then step **62** is entered, where the {SYN+GET} is forwarded to the next intermediate node on T.

However, if the next node is a home server **20**, then the step **63** is performed. In particular, snooper **28** establishes the server half of the TCP connection with the client **12**, and replaces the {SYN+GET} with a {PROXY_GET} request to the local resource manager **24**. The resource manager **24** translates the {PROXY_GET} request to an explicit {GET} issued to the home server **20**. The response of the home server **20** response is then relayed to the client **12** in the same manner as if the cache server was caching the requested document.

Another shortcoming of the caching technique described thus far is that the path along the tree T between a particular client **12** and the home server **20** can change after a leaf node server **38** or an intermediate node server **39** decides to service a request. This may occur, for example, when a network connection, or link, is lost between two server nodes. FIG. **7** shows this relatively rare case where the path between the client **12** and the home server **20** changes while an intermediate cache server **16*b*** is processing a document request from client **12**. All {ACK}s sent by the client **12** will now follow the new path, through a new cache server **16*x***, to the home server **20**. This causes cache server **16*b*** to time-out and retransmit its packets.

To solve this problem, the snooper **28** at server **16*b*** may keep track of the number of times a packet is re-transmitted. If a packet is re-transmitted more than a predetermined number of times, for example, three times, the snooper **28** then assumes that the path between the client **12** and the home server **20** has changed, and then takes steps to terminate the connection with the client **12**. In particular, the snooper **28** aborts the connection with the client **12** and aborts the connection with cache server **16*b***, simultaneously spoofing the home server **20** and sending a reset packet (i.e., an {RST} packet) to the client **12**.

In another approach the leaf node servers **28** closest to the clients **12** and the last hop nodes closest to the server **20** are provided with only one possible route to the clients **12** and servers **20**, respectively. This is accomplished by having the cache servers forward client request messages over cache server-to-cache server permanent TCP connections, instead of simply letting the request messages follow their normal routes. The set of connections, being implemented as a set of properly joined TCP connections, thus automatically adapts to any changes in IP routing as the network configuration changes.

3. Neighborhood Discovery

However, any resulting changes in the configuration of adjacent cache servers must also be detected by communication with neighboring cache servers in order to achieve resource load balancing and other advantages possible with the invention. In particular, each cache server **16** participating in the above-described scheme has to determine which other servers are in its neighborhood. In addition, on each routing tree T, a cache server **16** has to distinguish between upstream servers (located at parent nodes) and down stream servers (located at child nodes). A particular node, i, in the tree T is the parent of a node j, if i is the first cache server **16** on the route from j to the home server **20**, in which case node j is also referred to as the child of node i.

One method for a cache server **16** to discover its neighborhood requires some assistance from the underlying router **14** and snooper **28**. At selected times, the resource manager **24** asks the local router **14** to issue neighborhood discover messages to each destination in a routing table which the router **14** maintains.

These neighborhood discovery packets are then intercepted by a given snooper at another node having a cache server **16** in the tree. It is then responsibility of the intercepting cache server **16** to send a reply to the resource manager **24** at the cache server **16** that issued the neighborhood discover packet, announcing that it is a parent (e.g., that it is closer to the home server **20** than the issuing cache server) and the identity of the tree T that it is on. The destination port for neighborhood discover packets may be assigned an unlikely port number, to ensure that the destination home server **20** does not attempt to process un-intercepted neighborhood packets. A hop count field can also be used to limit neighborhood discover packets from excessive forwarding.

The main drawback of this approach is that it would flood the network with neighborhood discover packets. An alternative approach is to use document request message packets (i.e., the {SYN+GET} packets) that fly-by the filter in each cache server **16** anyway.

In this approach, each document request message contains a field identifying the previous hop, that becomes, under the scenario implemented above, an identification of the last cache server **16** that a particular request packet passed through.

As a request passes through a router **12** (i.e., it is not intercepted), the local snooper **28** stamps the IP address of the attached cache server **16**. When a cache server **16** wants to discover its neighborhood, it then instructs its attached snooper **28** to extract the last observed destination and last hop address from request packets and then passes this information up to the local resource manager **24**.

As shown in FIG. **8**, a typical HTTP {GET} message follows a path from the client **12** through A to the home server **20** and is intercepted by intermediate cache **16c**. While cache server **16c** is processing the request, the path between the home server **20** and the client **12** changes causing all acknowledgments to use a different path.

Using this information the resource manager **24** at cache server **16c** determines both which routing trees it is on and any down stream cache servers **16** on each tree. Once server **16c** determines that server **16b** is its downstream child on tree T, cache server **16c** has to explicitly inform cache server **16b** that it is its parent on T. To reduce the number of messages exchanged between the different components (snoopers **28** and resource managers **24**), the snoopers **28** can cache a number of packets and forward them all at once to the resource managers **24**.

Neighborhood information is maintained for a predetermined number, such as two, of neighborhood discovery epochs. If no requests are received through a child cache server **16b** during these periods, the child cache server **16b** is removed from the cache server **16c**'s model of the neighborhood. The parent cache server **16c** then also informs the child cache server **16b** of its intention to do so.

It is also possible that a cache server **16** does not have a parent snooper **28** on the routing tree to the home server **20**. In this case, the snooper **28** at cache server **16b** sends a neighborhood discovery packet in the direction of the home server **20**. An upstream snooper such as the one at server **16c** receives the packet and informs **16b** that it is its parent on the tree to the home server **20**. However, if the snooper **28** at **16b** does not have a parent node such as **16c** on the tree to home server **20** it replaces **16b** address on the neighborhood discovery packet and forwards it in the direction of the home server **20**.

This neighborhood discovery scheme has a number of advantages. First, the routing tree T does not have to be completely constructed for the caching protocol to start operating. Another advantage is that the cooperating cache servers **16** can dynamically discover and adapt to routing changes. Finally the protocol is totally distributed and is therefore robust against server failures.

4. Load Balancing

Unlike most other caching schemes, the caching scheme according to the invention requires distribution of cache copies to the cache servers **16** prior to clients actually requesting them. In other words, documents are moved among the cache servers **16** in anticipation of future document requests, rather than in direct response to any one particular document request message by the clients **12**.

The above scheme of document caching and neighborhood discovery lends itself to a number of different types of such cache load distribution and/or load balancing objectives for both the cache servers **16** as well as the communication paths which interconnect them. In the preferred embodiment, this load distribution scheme attempts to avoid introducing an overhead that grows quickly with the size of the caching system, by using a diffusion based caching algorithm that relies strictly on local information.

In particular, the resource managers **24** create cache copies only when an upstream ("parent") node in the routing tree T detects a less loaded downstream ("child") node or link, to which it can shift some of its document service load by giving it a copy of one of its cached documents. "Load" herein can be a number of different performance criteria such as rate of request fulfillment, client response time, or fraction of time the server is busy.

An imbalance in the opposite direction causes a child node to delete some of its cached documents, or to otherwise reduce the fraction of requests for these documents that it wishes to serve.

Typically, documents which are the one being requested of parent nodes most often by a child node according to some measure are the documents which are directed to less loaded child nodes.

Similarly, when a cache server **16** must choose to drop documents from its local cache store **18**, such documents are those typically being the least requested documents.

More particularly, when relegating load to a neighbor, a cache server **16** can push or release a document to a child or a parent, respectively. There are two goals with the present document caching by diffusion mechanism, which typically do not exist in traditional load diffusion processes. This is manifested in the need for the cache servers to determine which document to replicate, as well as to determine what fraction of document requests (i.e., load) should be relegated to an underloaded neighbor **30**.

A first goal is to determine how a cache server selects a document to pass to an underloaded neighbor. An objective here is to extract the maximum capacity of all of the cache servers **16** in the network. A second objective is to reduce response time, by moving popular documents closer to clients and less popular documents away from clients, all by creating the least number of replicas. These goals are accomplished while also considering communication path load between cache servers **16**.

To achieve these objectives, an overloaded cache server located at node i determines the least underloaded cache server at a child node j such that

$$L_j \leq L_k \forall k \in C_i$$

where L_i is the load at a particular cache server i, and Ci is the set of all cache servers. The overloaded cache server i pushes to the child j a fraction of the requests for the most popular document. Specifically, a document d is pushed from node i to node j if it has the following property

$$P_{ji}(d)=\max_{g \in D_i} P_{ji}(g)$$

where Di is the set of documents cached at node i. In the other direction, a document d is released by node i to node j if

$$P_{ji}(d)=\min_{g \in D_i} P_{ji}(g)$$

This policy is called max-min document diffusion, and it will satisfy the first two goals stated above.

A given cache can be viewed as a bin that is to be filled with documents in such a manner as to contribute the most to alleviating load. Thus, the larger the load associated with a given document, the smaller the number of documents that need to be placed in the particular cache. Max-min document diffusion provides a least number of documents in a cache to satisfy this third condition.

Having now determined which documents to diffuse, a cache server must also figure out how to change the fraction of requests that they intercept when relegating their load to a neighbor. This fraction can be determined by considering how a cache server pushes a fraction of the load imposed by a document. Specifically, assume that node i is the parent of node j and that i is overloaded with respect to j. Furthermore, assume that node i wishes to push a number of requests equal to $R_{ij}(d)$ of a total number of requests that it serves for document d to node j. If the total number of requests is defined as

$$\beta_{ji}(d)\Delta_{ji}(d)$$

then the number of requests serviced by node i for document d is therefore given by

$$P_{ii}(d)=\sum_{m\in C_i}\beta_{mi}(d)\Delta_{mi}(d)$$

and, after pushing $R_{ij}(d)$ requests to node j, node i will be serving a number of requests given by

$$P'_{ii}(d)=\sum_{m\in C_i}\beta_{mi}(d)\Delta_{mi}(d)-\beta_{ji}(d)\Delta_{ji}(d)+\beta'_{mi}(d)\Delta'_{mi}(d)$$

where

$$\Delta'_{ji}(d)=\Delta_{ji}(d)-R_{ij}(d)$$

and $\Delta_{ji}(d)$ represents the number of requests not intercepted by node j after the push to node j takes effect. Of $\Delta'_{ji}(d)$ the new fraction intercepted by node i is denoted by $\beta'_{ji}(d)$. Note also that

$$R_{ij}(d)=\beta_{ji}(d)-\beta'_{ji}(d)\Delta'_{ji}(d)$$

Using the above values of $\Delta'_{ji}(d)$ and $R'_{ji}(d)$ it is straightforward to see that

$$R_{ij}(d)=\beta_{ji}(d)\Delta_{ji}(d)-\beta'_{ji}(d)(\Delta_{ji}(d)-R_{ij}(d))$$

and by algebraic manipulation, that the new value of

$$\beta'_{ji}(d)=\frac{\beta_{ji}(d)\Delta_{ji}(d)-R_{ji}(d)}{\Delta_{ji}(d)-R_{ij}(d)}$$

After computing the new fraction $\beta'_{ji}(d)$, node i updates its associated filter and forwards a copy of document d to node j, directing node j to increase the number of requests that node j intercepts by $R_{ij}(d)$.

Once node j receives this information it computes

$$\beta'_{mj}(d)=\frac{R_{ij}}{\Delta_{ji}}(1-\beta_{mj}(d))+\beta_{mj}(d),\ \forall\, m\in C_j$$

and reflects these updates in its local filter code. To complete the analysis, note that the number of requests filtered by node j increases by $R_{ij}(d)$. It is also known that

$$P'_{jj}(d)=\sum_{m\in C_j}\beta'_{mj}(d)\Delta_{mj}(d)$$

and, by substituting the new value of $\beta'_{mj}(d)$ one obtains

$$P'_{jj}(d)=\sum_{m\in C_j}\left[\frac{R_{ij}(d)}{\Delta_{ji}(d)}(1-\beta_{mj}(d))+\beta_{mj}(d)\right]\Delta_{mj}(d)$$

and by algebraic manipulation, that

$$P'_{jj}(d)=\frac{R_{ij}(d)}{\Delta_{ji}(d)}\Delta_{ji}(d)+P_{jj}(d)$$
$$=P_{jj}(d)+R_{jj}(d)$$

thus completing the analysis.

With large documents, it may be advantageous for the local cache **18** to only include a portion of the requested document. In this event, the cache server can begin to provide the locally cached portion of the document to the client **12**, while at the same time requesting the remainder of the document be sent by the home server **20**.

Other criteria may include document size, document type, direction of transmission, or priority associated with particular documents.

The cache server **16** and router **12** may also make use of communication buffers in order to favor message traffic which is determined to have higher priority.

Related documents of the same types or which are recognized in some way by cache servers **16** as normally being requested together in close succession can be shifted between cache servers in the same operation.

Each cache server **16** is thus given the ability to cache and discard documents based upon its local load, its neighbors' loads, communication path load, and on document attributes such as popularity, size, cost to fetch, etc. In particular, each server maintains an estimate of the load at its neighbors and/or also transmits at various times, or "gossips" about its actual load to neighboring cache servers **16**. If a cache server **16** notices that it is overloaded in some respect as compared to its neighbors, it relegates a fraction of its future predicted work to its less loaded child or parent neighbors as described above.

The invention thus also lends itself to load splitting, where neighbor ("sibling") nodes on the same path in the routing tree T for a given home server may share the request load for a particular document.

The above document diffusion algorithm may experience certain difficulties in optimized load distribution. In particular, a given cache server j can become a potential barrier to cache copies when it has at least two child cache servers k and k', and a parent cache server i, such that the load, L, on each server satisfies the expression:

$$L_{k'}\leq L_j\leq L_i\leq L_k$$

and cache server j does not cache any of the files required by its under-loaded child k.

FIG. 9(*a*) illustrates an example of such a situation. The caching system consists of a home server 20 (at node number 1) and three intermediate servers 39 (as nodes 2, 3, and 4.) Requests are only being generated by the leaf nodes, in particular, documents d_1 and d_2 are being requested by node 4, and d_3 is being requested by node 3. The figure shows the placement of cache copies at particular nodes and the requests serviced by each cached copy.

In this example, the cache server 16 at node 2 is the potential barrier. It cannot diffuse any load to the cache server at node 3, since it does not cache d_3. In addition, the cache server 16 at node 2 isolates the cache server at node 1 from recognizing the existence of the problem.

One possible optimized load assignment would distribute the load evenly among all four nodes with each node servicing requests. FIG. 9(*b*) illustrates file cache and load distributions that would satisfy this condition.

The diffusion based scheme can be altered, however, so that a cache server 16 can detect such undesirable states as shown in FIG. 9(*a*) and recover from them. In particular, a cache server k can assume that its parent server j is a potential barrier if k remains under-loaded, relative to j , for more than a predetermined number of time epochs, such as two, without action taken by j to correct the under-loaded situation. In the example of FIG. 9(*a*), cache server k would correspond to node 3, with cache server j corresponding to node 2. Upon detecting a lack of diffused load from its parent node k, the child node can infer that the parent node does not cache any of the documents requested by the subtree rooted at k. Once copies of these documents are eventually served to it, the server k can then cache them normally. This technique is referred to as tunneling, because server k in this case is able to obtain and cache copies even in the presence of a parent node j which represents a high load barrier.

5. Other Features

The invention can be used to implement additional functionality in a network. These various functions are a direct result of the fact that the filter 26 located at the routers 14 can do more than simply divert requests for copies of documents to the local cache server 16.

For example, document popularity statistics are of necessity collected by the cache servers 16, in order to control the choice of which document to cache and in which cache server. The cache server 16 at each node keeps track of how many references to particular documents are coming in from where and knows an aggregate request amount.

This data can be collected at a central location, and arranged by groups of cache servers 16, such as by groups of cache servers 16 that are located in a particular network region. By collecting data in this fashion, it is then possible for the publisher of a document to not only obtain statistics about the "hit rate" on their material by specific clients, but also in which sub-portion of the network 10 the hits are coming from.

The invention also enables a type of accelerated distribution of documents into-the network 10 in anticipation of demand.

The cache servers 16 can also be used to host replicas of databases, search index files, and-other popular documents by acting as load splitters from the service provider perspective. In other words, database providers can arrange to have their documents placed into the network 10, pushing out data closer to the clients 12 that desire access to it, wherever these placements might be.

The cache servers may also implement authentication of the sources of request messages and other information. This can be done because the cache servers 16 automatically maintain information as to the client 12 which was the source of a particular document request message. If the client 12 is not among the authorized requesters for the document, the request message can be terminated at the cache server 16 before it even reaches the home server 20.

Selective security can also be provided for as well. The mechanism arises from the fact that the nodes each have a filter 26, a resource manager 24, a cache repository 18, and an HTTP proxy 22. The filter 26 may be used not only to keep track of how to redirect requests for cache copies to the HTTP proxy 22, but may also restrict access to the cache copies, such as by authenticating the request for the information. As long as the first and last hop along the path from the client 12 to the server 20 are trusted, since the links between the caches servers 16 can easily be arranged to be trusted links, a secure link can be provided between the clients 12 and the home server 20 via the cache servers 16.

More generically, the cache servers 16 may transform documents at several points during their distribution by the home server 20 and subsequent delivery to clients 12. Such transformations may enable and/or implement several features that add value to caching a document or program, including such features as compression or encryption of documents and/or programs. Furthermore, any known security techniques can be applied as transformations at the source, destination or both (in the case of a transaction). In addition to encryption and decryption, these include authentication, authorization (access control), non-repudiation, and integrity control. These security techniques can use cryptographic techniques that usually require a key, and optionally use the physical path to a cache server to strengthen authentication, authorization, and non-repudiation.

In typical applications, these transformations will be matched (e.g., for every encryption there should be a matching decryption) and used individually. However, they may also be composed at different points in the lifetime of a document. For example, a document may be compressed, encrypted, decrypted, and decompressed, all at different points.

FIG. 10 illustrates one possible approach for applying transformations to documents. In this example, distinct transformations may occur at four different times as a document is distributed by the home server 20 and delivered to the client 12:

at time T1, as the request is made by the client 12

at time T2, as the request is forwarded to server 20

at time T3, as the reply is sent by the server 20

at time T4, as the reply is forwarded to the client 12

Note that even though the transformations in FIG. 10 are associated with four phases of communication, in reality, transformations are performed by the nodes themselves. Thus, T1 may be performed on the request by cache server 16-6 after it is received, or by client 12 before it is sent. In the former case, cache server 16-6 is performing the transformation. In the later case, the client 12 is performing the transformation, and merely tunneling this through cache server 16-6. Likewise, as the request is forwarded to the home server 20 the transformation T2 is performed by cache server 16-7 or home server 20. The same alternatives apply for T3 and T4, as the reply is sent from home server 20 and ultimately forwarded to client 12.

Where the transformations as performed in FIG. 10 have an important role is in key placement for security (or any

other transformations which requires input other than the document). If the cache servers **16** themselves are implementing security, cryptographic keys must be distributed to the cache servers **16**, as needed. In this case, end-to-end security can be provided by adding a secure channel between client **12** and cache server **16-6** as well as between cache server **16-7** and server home **20**. If the client **12** and home server **20** implement their own end-to-end security the cache servers **16** do not hinder, and possibly cooperate in, distributing keys to the client **12** and home server **20**.

FIG. **10** shows the cache servers **16-6** and **16-7** at the edges of the network, whereby there is a direct path between client **12** and cache server **16-6** as well as between cache server **16-7** and home server **20**. However, the design as described does not preclude a more general configuration, when transformations are performed by intermediate cache servers **16-9** and **16-10**. In other words, in FIG. **10** a transformation may be made at an intermediate cache server **16-9, 16-10** during the request, or reply, or both.

The scheme also allows for a form of transparent document compression, which is another form of conserving bandwidth, as long as a particular home server **20** and client **12** are committed to communicating by using cache servers **16**. In particular, after the first hop along the path between the client **12** and the server **20**, the first leaf node server **16-6** can implement compression or encryption in a manner which is transparent to both the client **12** and the home server **20**.

Documents can also be virus-scanned or code-certified prior to being forwarded.

The invention also permits the efficient caching of dynamic content documents, where the end result of which data is actually returned to the client changes on the fly. Such documents may include, for example, documents having embedded Java code.

The cache servers **16** can accomplish efficient distribution of such dynamic content documents by caching not only the data for the document, but by also caching the programs that specify how the document is to be displayed when the data is retrieved. If the programs are of the type that are normally executed at the client **12** at the time of display of the documents, then the programs and the data are simply transferred to the client **12**.

If, however, the programs are of the type which the client **12** expects will be running on the home server **20**, the cache server **16** performs an additional level of home server spoofing by also running the programs. In this instance, it may be necessary for the cache server **16** to maintain an interactive session state with the client **12** in order to complete the spoofing of the home server **20**.

The invention also inherently improves the delivery of stored media such as audio and video data files since number of hops between a home server **20** and a client **12** are reduced.

While we have shown and described several embodiments in accordance with the present invention, it is to be understood that the invention is not limited thereto, but is susceptible to numerous changes and modifications as known to a person skilled in the art and we therefore do not wish to be limited to the details shown and described herein but intend to cover all such changes and modifications as are obvious to one of ordinary skill in the art.

What is claimed is:

1. In a system containing a plurality of computers which communicate over a network using communication protocols, with the computers at certain nodes in the network acting as home servers for storing information in the form of documents, and with certain other computers acting as clients that send document request messages to the servers at an application layer, the document requests message being requests for documents stored at the home servers, a method of fulfilling document request messages comprising the steps of:

(a) storing local cache copies of documents at a plurality of intermediate node locations in the network; and

(b) in response to a particular one of the clients generating a particular application layer document request message addressed to a particular one of the home servers, attempting to fulfill the particular application layer document request message at one of the intermediate node locations by, at a selected communication layer lower than the application layer,

(i) intercepting the document request message at an intermediate node location through which the document request naturally travels, the intermediate node being located along a path through the network from the client towards the home server, the path being a route from the client to the home server as determined by the home server address, and corresponding to an identical path that the message would travel through the network in the absence of any cache copies being stored at intermediate node locations; and

(ii) returning one of the local cache copies to the application layer at the client, such that the application layer request message is intercepted by the lower layer at the intermediate node and such that the application layer on the server does not receive application layer document request message.

2. A method as in claim **1** additionally comprising the step of, at selected intermediate nodes:

(c) if a particular document request message cannot be fulfilled by providing one of the local cache copies, forwarding the request message towards the home server along the path by continuing to address the request message to the particular home server.

3. A method as in claim **1** wherein the step of fulfilling the particular document request additionally comprises, in at least one intermediate node, wherein the intermediate node is neither a client nor a home server, the step of:

(c) storing local cache copies of particular documents which are also stored on other intermediate nodes;

(d) determining the identity of a neighboring intermediate node that stores local cache copies; and

(e) allocating the fulfillment of document request messages among the intermediate node and the neighboring intermediate node based upon the availability of document request message load fulfillment resources at the intermediate node.

4. A method as in claim **3** wherein the message load fulfillment resources include the caches.

5. A method as in claim **3** wherein the message load fulfillment resources include communication path load.

6. A method as in claim **3** in which the step of allocating the fulfillment of document request messages additionally comprises the step of:

(f) exchanging status messages between the intermediate node and the neighboring node, the status messages including information selected from at least one of processing load, communication path load, document size, document request message response time, document request message request rate, document request message rate of change, or home server operability.

7. A method as in claim 1 wherein the step of fulfilling the particular document request additionally performs communication path load distribution between paths that interconnect the nodes, by further comprising the steps of:

(c) storing local cache copies of particular documents which are also stored on other nodes;

(d) determining the identity of a neighboring node that stores local cache copies; and

(e) allocating the fulfillment of document request messages among the intermediate node and the neighboring intermediate node based upon the communication path load.

8. A method as in claim 2 additionally comprising the steps of, at selected intermediate nodes:

(d) determining an identity of a first neighboring node from which a particular document request message is received; and

(e) determining an identity of a second neighboring node to which a particular document request message is routed on the path to the home server if the particular document request message cannot be fulfilled by returning the local cache copy to the client.

9. A method as in claim 8 wherein the step of storing local cache copies further comprises the step of:

(f) determining a node state parameter at the first neighboring node and a node state parameter at the local node, and when the first neighboring node and local node state parameters are different by a predetermined amount from one another, forwarding a copy of at least one of the local cache copies to the first neighboring node for storing at the first neighboring node location.

10. A method as in claim 9 wherein the node state parameters are selected from the group consisting of rate of request fulfillment, change in rate of request fulfillment, document size, document fetch response time, communication path load, cache server load, or home server operational status.

11. A method as in claim 8 wherein the step of storing local cache copies further comprises the step of:

(f) determining a node state parameter at the second neighboring node and a node state parameter at a local node, and when the node state parameters at the second neighboring node and local node differ by a predetermined amount, deleting at least one of the local cache copies.

12. A method as in claim 11 wherein the node state parameters are selected from the group consisting of rate of request fulfillment, change in rate of request fulfillment, document size, document fetch response time, communication path load, cache server load, or home server operational status.

13. A method as in claim 8 wherein the step of storing local cache copies further comprises the step of:

(f) determining a node state parameter at the second neighboring node and a node state parameter at a local node, and when the node state parameters at the second neighboring node and local node different by a predetermined amount, in the step of fulfilling, reducing a proportion of the document requests that are fulfilled by providing local cache copies to the client.

14. A method as in claim 1 wherein the step of fulfilling additionally comprises the step of:

(c) returning the local cache copy to the client if a local node state parameter differs from a predetermined amount.

15. A method as in claim 1 additionally comprising the step of, at the selected intermediate nodes:

(c) recording load statistics as to the number of request messages fulfilled by providing one of the local cache copies to the client.

16. A method as in claim 1 additionally comprising the step of, at the selected intermediate nodes:

(c) recording request message statistics as to the number of request messages received and fulfilled for a particular document.

17. A method as in claim 1 additionally comprising the step of, at the selected intermediate nodes:

(c) recording response time statistics as to the number of request messages received for documents stored at a particular home server.

18. A method as in claim 1 wherein document request messages are received at the intermediate nodes on a plurality of communication paths, and the method additionally comprises the step of, at the selected intermediate nodes:

(c) recording request message statistics to track which document request messages are received from a particular path.

19. A method as in claim 1 wherein document request messages are received at the intermediate nodes on a plurality of communication paths, and the method additionally forecasts communication path usage by criteria selected from one of overall network demand, network regional demand, or by specific client demand.

20. A method as in claim 1 wherein the step of fulfilling document request messages additionally comprises the step of:

(c) authenticating a node over which a document request message arrives from a client, prior to forwarding the document request message.

21. A method as in claim 1 wherein the step of fulfilling document request messages additionally comprises the step of:

(c) authenticating a node from which the cached document originated prior to returning the local cache copy to the client.

22. A method as in claim 1 wherein the home servers also store programs for operating on the documents, and step (b) additionally comprises the step of, at the intermediate node:

(c) storing local cache copies of selected programs as certain selected programs related to requested document copies are obtained from the home servers.

23. A method as in claim 22 additionally comprising the step of:

(d) executing the local cache copies of selected programs at the intermediate nodes upon demand from the client.

24. A method as in claim 22 additionally comprising the step of:

(d) maintaining an interactive session between the intermediate node and the client such that the intermediate node acts as the home server would act in the event that the home server had fulfilled the document request message.

25. (Amended) A method as in claim 1 wherein the step of fulfilling the document request message additionally comprising the step of:

(c) applying selected programs to the local cache copies to obtain results, prior to returning the results of applying the programs to the client.

26. A method as in claim 1 wherein the documents are selected from the group of multimedia documents, programs, data bases, compressed data, or encrypted data.

27. A method as in claim 8 wherein multiple intermediate nodes are located between the home server and the client,

and wherein a particular document is pushed into the network by routing it to a plurality of nodes depending upon an expected rate of demand upon the document.

28. A method as in claim **9** wherein the predetermined rate of request fulfillment depends upon document attributes selected from the group of expected rate of request fulfillment, expected change in rate of request fulfillment, document size, expected document fetch response time, expected communication path load, or expected cache server load.

29. A method as in claim 1 additionally comprising the step of, at the intermediate nodes,

(c) storing, with the local cache copies, data indicating a condition as to the release of the local cache copies; and

(d) returning the local cache copy to the client only when the condition is satisfied.

30. A method as in claim **29** wherein the condition is a time of release of the document.

31. A method as in claim 1 wherein the step of storing cache copies is selectively executed based upon predetermined conditional criteria.

32. A method as in claim **31** wherein the predetermined conditional criteria include time of day.

33. A method as in claim **31** wherein the predetermined conditional criteria is at least one selected from the group of document size, desired document request message response time, document request message rate, rate of change of the document request message rate, cache server load, communication path load, or home server operational status.

34. A method as in claim 1 wherein the step of storing cache copies additionally comprises the step of storing a partial copy of a document which is larger than a predetermined size.

35. A method as in claim 1 additionally comprising the step of:

(c) deleting cache copies of documents selectively executed based upon predetermined conditional criteria.

36. A method as in claim **35** wherein the predetermined conditional criteria include time of day.

37. A method as in claim **35** wherein the predetermined conditional criteria is at least one selected from the group of document size, desired document request message response time, document request message rate, rate of change of the document request message rate, cache server load, or communication path load.

38. A method as in claim 1 additionally comprising the steps of, to maintain cache consistency:

(c) at the home server, attaching a document expiration time stamp to the document;

(d) at an intermediate node, examining the time stamps attached to a cached document to determine if the document has expired; and

(e) if the cached document has expired, either deleting or refreshing it.

39. A method as in claim 1 additionally comprising the steps of:

(c) at the home server, attaching a document modification time stamp to the document; and

(d) at an intermediate node, estimating the modification rate of the document, and if the document modification rate is greater than the request rate by a predetermined amount, deleting the cache copy, and if the modification rate is less than the request rate by a predetermined amount, requesting an updated cache copy.

40. In a system containing a plurality of computers which communicate over a network using a layered communication protocol, with the computers at certain nodes in the network acting as home servers for storing information in the form of documents, and with certain other computers acting as clients that send document request messages to the servers at an application layer level, the document request messages being requests for documents stored at the home servers, a method of fulfilling document request messages by transparent proxying comprising the steps of:

(a) storing local cache copies of documents at a plurality of intermediate node locations in the network; and

(b) in response to a particular one of the clients generating a particular application layer document request message intended to be sent to a particular one of the home servers, fulfilling the particular application layer document request message at one of the intermediate node locations along a path from the client to the home server through which the document request would naturally flow in the absence of any cache servers, by intercepting the document request message and returning one of the local cache copies in a response message to the application layer at the client, such that the application layer request message is intercepted at the intermediate node and such that a network connection is not established with the application layer on the home server and such that the addresses in a response message are the same as if the home server had fulfilled the document request message.

41. A method as in claim **40** wherein an intermediate node comprises a cache server and a router, additionally comprising the steps of, at the router:

(c) recognizing document request messages that are to in be intercepted, and extracting such messages for processing by the cache server.

42. A method as in claim 1 additionally comprising the steps of:

(c) simultaneously pre-fetching related documents together, wherein related documents are those documents that are most frequently requested in close succession.

43. A method as in claim 1 additionally comprising the steps of: at the client,

(c) generating a connection request message which requests a communication path to be established between the client and a home server; and at an intermediate node,

(d) upon receiving a connection request message from the client, waiting for the receipt of a document request message, and forwarding the connection request message and the document request message together addressed to the home server.

44. A method as in claim **43** additionally comprising the steps of:

at an intermediate node which caches the document indicated by the request message,

(e) acknowledging the connection request to the client, and returning the requested document to the client.

45. A method as in claim 1 additionally comprising the step of:

(c) operating a cache server which selects documents to store in a local cache and documents to remove from the local cache, such that the cache server selects a most often requested document to replicate at a neighboring cache server located at a node in the path, and chooses the least often requested documents to drop from its own memory.

46. A method as in claim 1 wherein each intermediate node includes a resource manager associated with it, and the resource manager performs the steps of:

(c) allocating the use of communication buffers to buffer incoming and outgoing messages so as to favor message traffic that is designated as more important.

47. A method as in claim 46 comprising the steps of:

(d) allocating the use of the communication buffers and communication path bandwidth based upon content attributes selected from the set of document size, document type, direction of transmission, or priority.

48. A method as in claim 1 additionally comprising the steps of, at the intermediate nodes,

(c) comparing a node state parameter at the intermediate node with a node state parameter at a neighboring cache server; and

(d) if the node state parameters at the neighboring cache server and the intermediate node differ by a predetermined amount for more than a predetermined period of time, at the intermediate node, inferring that the neighboring cache server does not cache one or more documents being served by the intermediate node.

49. A method as in claim 48 wherein the node state parameters are selected from the group consisting of rate of request fulfillment, change in rate of request fulfillment, document size, document fetch response time, communication path load, home server operability, or cache server load.

50. In a system containing a plurality of computers which communicate over a network, with certain computers acting as home servers for storing information in the form of documents, and with certain other computers acting as clients that send messages which are requests for documents stored at the home servers, a method of fulfilling document request messages by distributed caching comprising the steps of:

(a) routing document request messages from a client to a home server along a path from the client to the home server, the path comprising a plurality of hops including a plurality of intermediate node locations in the network through which the document request would naturally flow through in the network in the absence of cache servers;

(b) at selected intermediate nodes,

(i) storing local cache copies of selected documents at intermediate nodes;

(ii) filtering document request messages to determine if a particular document request message made by a client can be fulfilled by providing one of the local cache copies to the client, and if so, returning the local cache copy to the client; and

(iii) if not possible to so fulfill the document request message forwarding the document request message to the home server along the path which the document request message would naturally follow through the network.

51. In a system containing a plurality of computers which communicate over a network, with the computers at certain nodes in the network acting as home servers for storing information in the form of documents, and with certain other computers acting as clients that send document request messages which are requests for documents stored at the home servers, a method of fulfilling document request messages at node through which the document request messages travel comprising the steps of:

(a) storing local cache copies of documents at a plurality of intermediate node locations in the network;

(b) in response to a particular one of the clients generating a particular document request message intended to be sent to a particular one of the home servers;

(i) determining a path for the particular document request message to travel from the particular client to the particular home server along a path of nodes located in the network between the particular client and the particular home server, the path being determined by an address of the home server and the path comprising a plurality of the intermediate node locations in the network;

(ii) forwarding the request message along the path as determined by the home server address in the request message; and

(iii) fulfilling the particular document request message at one of the plurality of intermediate node locations in the determined path of nodes by the steps of, if a local cache copy is available at the intermediate node, returning one of the local cache copies corresponding to a document specified in the particular document request message, and otherwise if a local cache copy is not available at the intermediate node, forwarding the request message in the direction of the home server along the path.

52. A method as in claim 51 additionally comprising the step of:

(c) at the intermediate node locations, coordinating the step of storing cache copies and splitting document fulfillment among the intermediate node locations to distribute document request load on the intermediate node cache servers or home server.

53. A method as in claim 51 additionally comprising the step of:

(c) at the intermediate node locations, coordinating the step of storing cache copies and splitting document fulfillment among the intermediate node locations to distribute communication path load.

54. A method as in claim 51 wherein the step of coordinating the step of storing cache copies among the intermediate node locations to reduce document load comprises the steps of:

(c) at neighboring node locations, replicating documents.

55. A method as in claim 51 wherein the step of storing copies of documents at intermediate node locations additionally comprises the step of:

(c) alternatively making such copies available or not available for fulfilling document requests based upon predetermined criteria.

56. A method as in claim 55 wherein the predetermined criteria is time of day.

57. A method as in claim 51 wherein the step of fulfilling the particular document request message at one of the intermediate servers additionally comprises the steps of:

(c) if the leaf node does not store the requested document opening a communication connection between the leaf node and the next intermediate node by sending a connection request message addressed to the home server that is intercepted by the next intermediate node server, the leaf node being one of the intermediate node locations that initially receives the document request message from the client; and

(d) forwarding the document request and the communication connection to another intermediate node that is located closer to the home server in the network than the leaf node.

58. A method as in claim 51 wherein the step of fulfilling the particular document request message at one of the intermediate servers additionally comprises the steps of:

(c) opening a communication connection between a leaf node and the home server, the leaf node being one of

the intermediate node locations that initially receives the document request message from the client; and

(d) relaying the document request to another intermediate node that is located closer to the home server in the network than the leaf node.

59. A method as in claim **51** wherein the cache servers additionally perform the steps of:

(c) acting as a communication proxy for the home server from the perspective of the client.

60. A method as in claim **50** wherein the step of filtering document request messages additionally controls access to documents, and the method additionally comprises the step of:

(c) filtering document request messages based upon a Uniform Resource Locator (URL) field in the document request message.

61. A method as in claim **50** wherein the step of filtering document request messages additionally controls access to documents, and the method additionally comprises the step of:

(c) filtering document request messages based upon an authentication field in the document request message.

62. A method as in claim **51** wherein the step of returning the cache copy of the document to the client additionally comprises the step of:

(c) virus scanning the document.

63. A method as in claim **51** wherein the step of returning the cache copy of the document to the client additionally comprises the step of:

(c) code certifying the document.

64. A method as in claim **51** wherein the step of returning the cache copy of the document to the client additionally comprises the step of:

(c) decoding the document.

65. A method as in claim **51** wherein the step of returning the cache copy of the document to the client additionally comprises the step of:

(c) controlling access to the cache copy of the document based upon client authentication and home server request.

66. A method of fulfilling requests for information in a network of computers, with at least some of the computers in the network acting as home servers for storing information, and at least some of the computers acting as clients that send request messages to the home servers, the method comprising the steps of:

(a) distributing cache copies of the information through the network by storing copies of the information in a plurality of computers in the network that act as cache servers;

(b) routing request messages from the client to the home server along a path consisting of a plurality of intermediate computers in the network, the path including computers through which the request messages travel in the absence of any cache servers with at least some of the intermediate computers also acting as cache servers, the request messages initiated by a particular one of the clients to obtain information from a particular one of the home servers; and

(c) transparently processing request messages as they are routed through the cache servers, such that request messages that can serviced by the cache servers instead of the home servers are serviced by the cache servers, in a manner which is transparent to the clients and the home servers.

67. A method as in claim **57** additionally comprising the step of:

(d) automatically moving cache copies among the cache servers in the network in response to predetermined criteria concerning the servicing of the request messages by the cache servers.

68. A method as in claim **1** wherein the multimedia documents contain digitized information selected from the group of text, graphics, audio, video, programs, or other data.

69. A method as in claim **1** wherein the network comprises a wireless network.

70. A method as in claim **1** wherein the intermediate nodes also perform the step of:

determining if a particular communication entity has failed, and if so, notifying another network entity.

71. A method as in claim **70** wherein the communication entity is one of a router, communication path, or communication path.

72. A method as in claim **70** wherein the other network entity is one of another intermediate node, a network administrator, or other networked computer system.

73. A method as in claim **55** wherein the predetermined criteria is a notification by the client.

74. A method as in claim **55** wherein the predetermined criteria is a notification by the home server.

75. A method as in claim **2** additionally comprising the steps of, at selected intermediate nodes:

(d) determining an identity of a first neighboring node from which a particular document request message is received; and

(e) determining an identity of a second neighboring node to which a particular document request message is routed on the path to the home server;

(f) if the particular document request message cannot be fulfilled satisfactorily by the intermediate node because of failed or slow processing at the intermediate node, altering the path to bypass the intermediate node.

* * * * *

US 20010003823A1

(19) United States
(12) Patent Application Publication (10) Pub. No.: US 2001/0003823 A1
MIGHDOLL et al. (43) Pub. Date: Jun. 14, 2001

(54) METHOD FOR DOWNLOADING A WEB PAGE TO A CLIENT FOR EFFICIENT DISPLAY ON A TELEVISION SCREEN

(76) Inventors: LEE S. MIGHDOLL, SAN FRANCISCO, CA (US); BRUCE A. LEAK, PORTOLA VALLEY, CA (US); STEPHEN G. PERLMAN, MOUNTAIN VIEW, CA (US); PHILLIP Y. GOLDMAN, LOS ALTOS, CA (US)

Correspondence Address:
WORKMAN NYDEGGER & SEELEY
1000 EAGLE GATE TOWER
60 EAST SOUTH TEMPLE
SALT LAKE CITY, UT 84111 (US)

(*) Notice: This is a publication of a continued prosecution application (CPA) filed under 37 CFR 1.53(d).

(21) Appl. No.: 09/095,457

(22) Filed: Jun. 10, 1998

Related U.S. Application Data

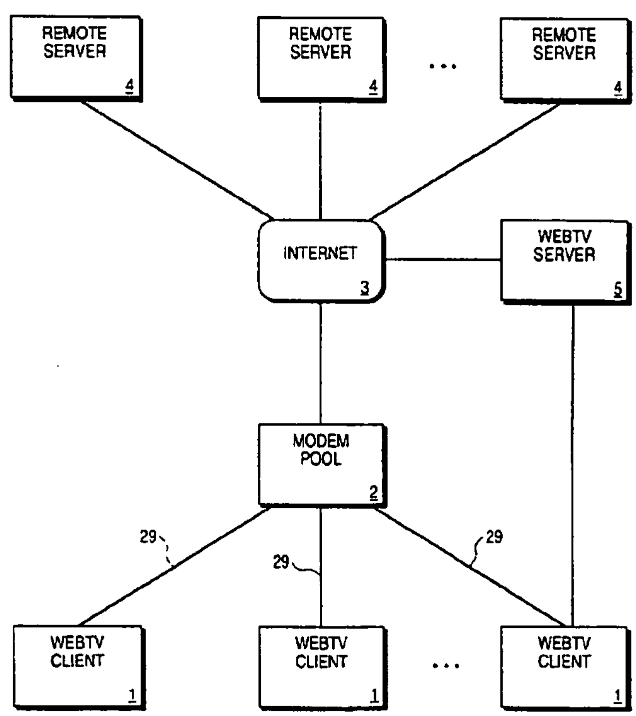
(63) Continuation-in-part of application No. 08/656,924, filed on Jun. 3, 1996, now Pat. No. 5,918,013, which is a continuation-in-part of application No. 08/660,087, filed on Jun. 3, 1996, now Pat. No. 5,896,444, which is a continuation-in-part of application No. 08/660,088, filed on Jun. 3, 1996, now Pat. No. 6,034,689.

Publication Classification

(51) Int. Cl.[7] G06F 15/16
(52) U.S. Cl. 709/200

(57) ABSTRACT

An improved method of providing a document to a client coupled to a server. The server provides a number of Internet services to the client, including functioning as a caching proxy on behalf of the client for purposes of accessing the World Wide Web. The proxying server retrieves from a remote server in response to a request from the client a Web document used to generate a Web page on a television screen coupled to the client. Prior to downloading the requested Web page to the client, the server lays out the entire Web page and separates the Web page into partitions such that each one of the partitions corresponds to the viewable display area of the television screen coupled to the client. The server downloads the HTML data that drives the layout within the viewable display area of the television screen. The server then downloads all of the image data that is displayed within the viewable display area of the television screen such that the portion of the Web page within the viewable area of the television can be fully generated and displayed by the client in a reduced amount of time. The server subsequently downloads the remaining partitions of the Web page in similar fashion until the entire web page has been downloaded to the client.



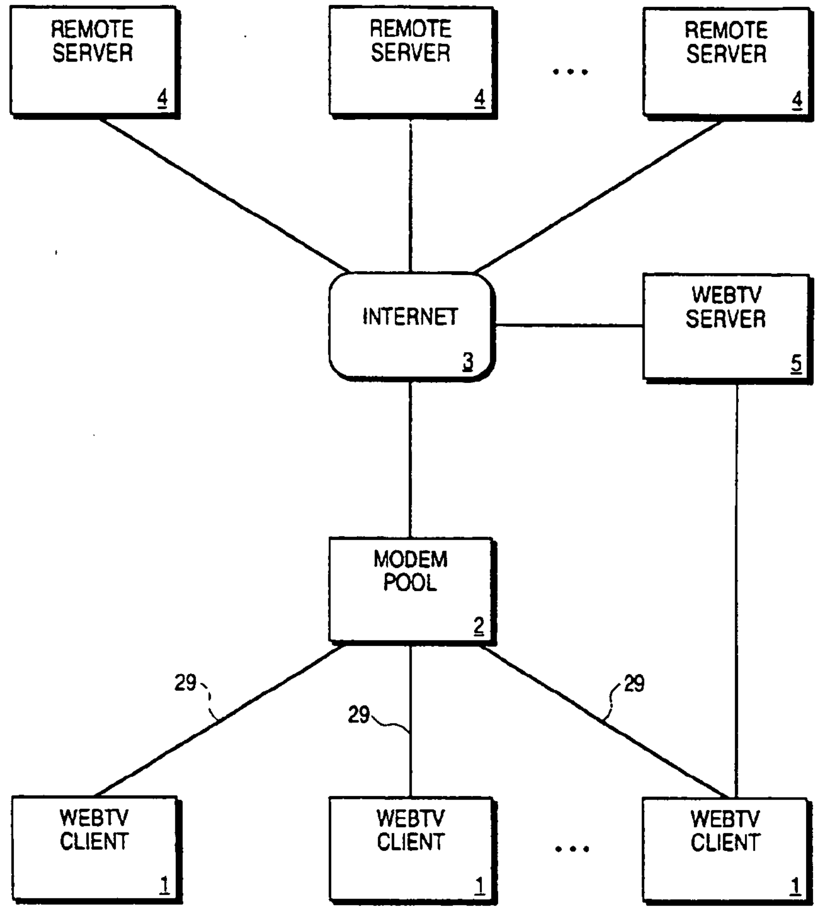

REMOTE SERVER 4
REMOTE SERVER 4
REMOTE SERVER 4
INTERNET 3
WEBTV SERVER 5
MODEM POOL 2
29
29
29
WEBTV CLIENT 1
WEBTV CLIENT 1
WEBTV CLIENT 1

FIG. 1

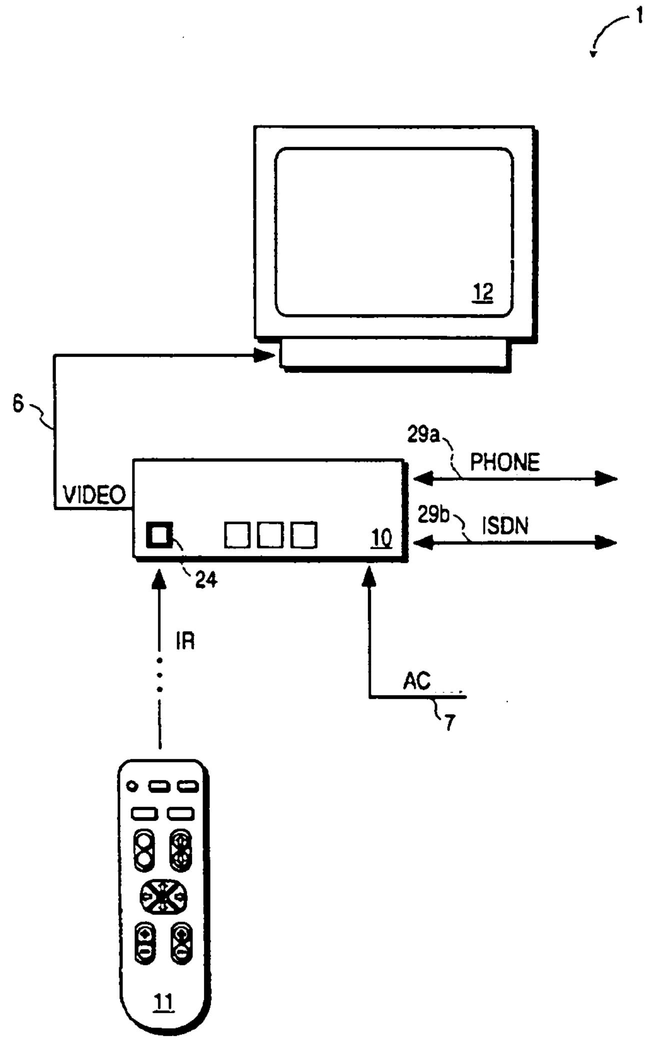
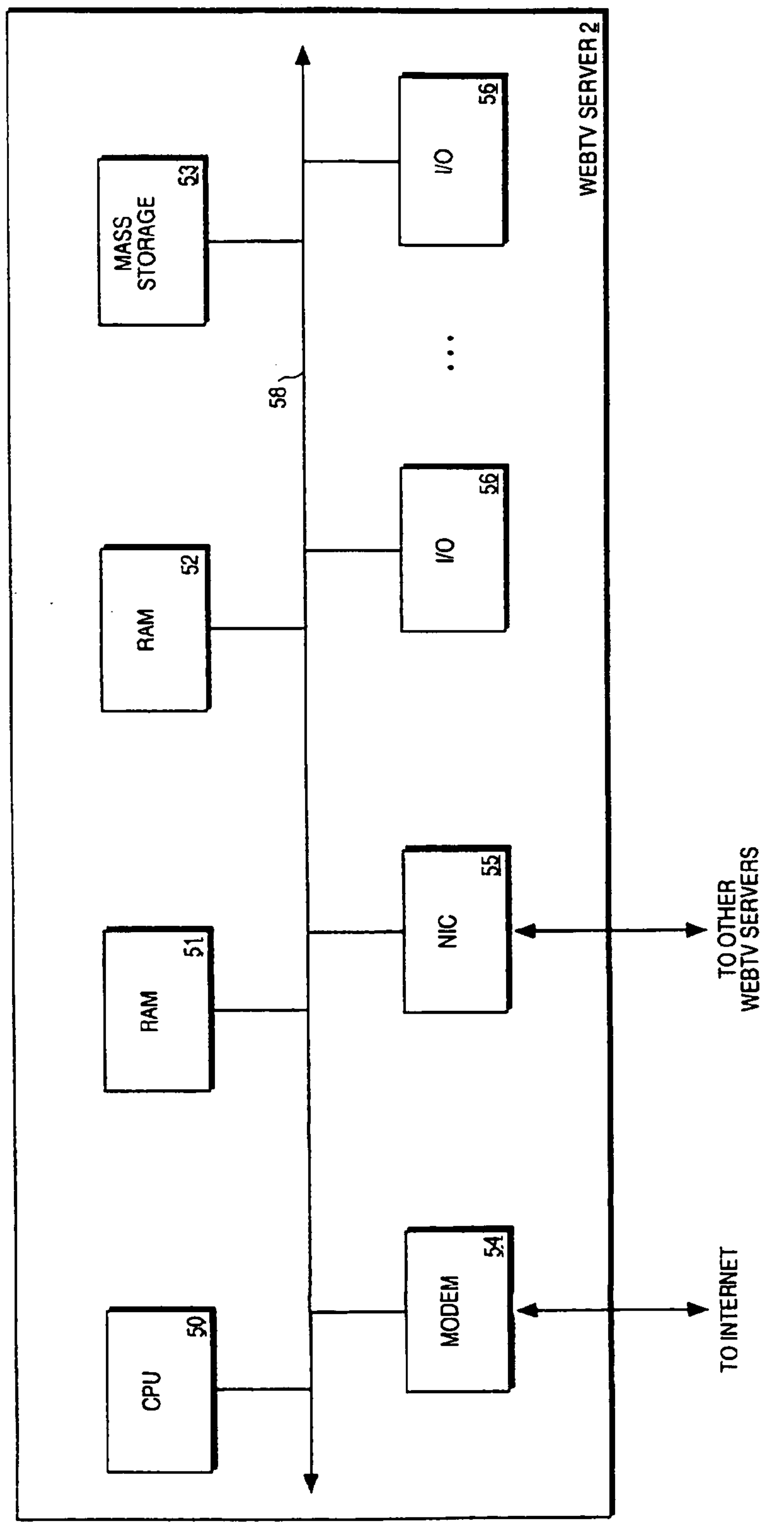


FIG. 2



WEBTV SERVER 2
MASS STORAGE
53
RAM
52
RAM
51
CPU
50
58
I/O
56
I/O
56
NIC
55
MODEM
54
TO OTHER WEBTV SERVERS
TO INTERNET

FIG. 3

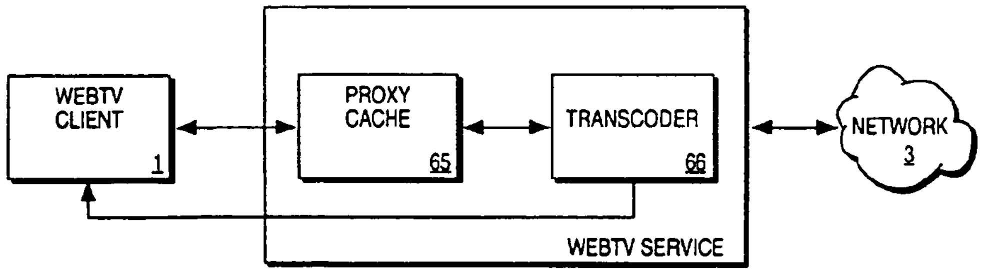

WEBTV CLIENT
1
PROXY CACHE
65
TRANSCODER
66
NETWORK
3
WEBTV SERVICE

FIG. 4A

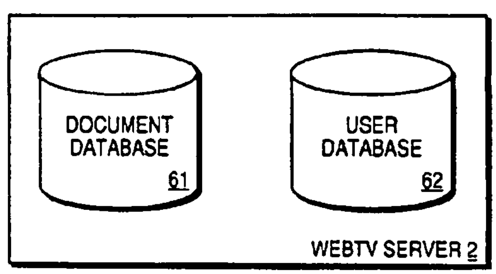

DOCUMENT DATABASE
61
USER DATABASE
62
WEBTV SERVER 2

FIG. 4B

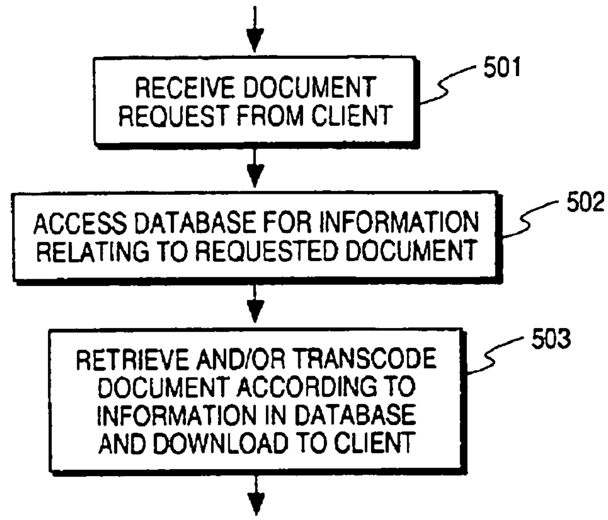

RECEIVE DOCUMENT REQUEST FROM CLIENT
501
ACCESS DATABASE FOR INFORMATION RELATING TO REQUESTED DOCUMENT
502
RETRIEVE AND/OR TRANSCODE DOCUMENT ACCORDING TO INFORMATION IN DATABASE AND DOWNLOAD TO CLIENT
503

FIG. 5

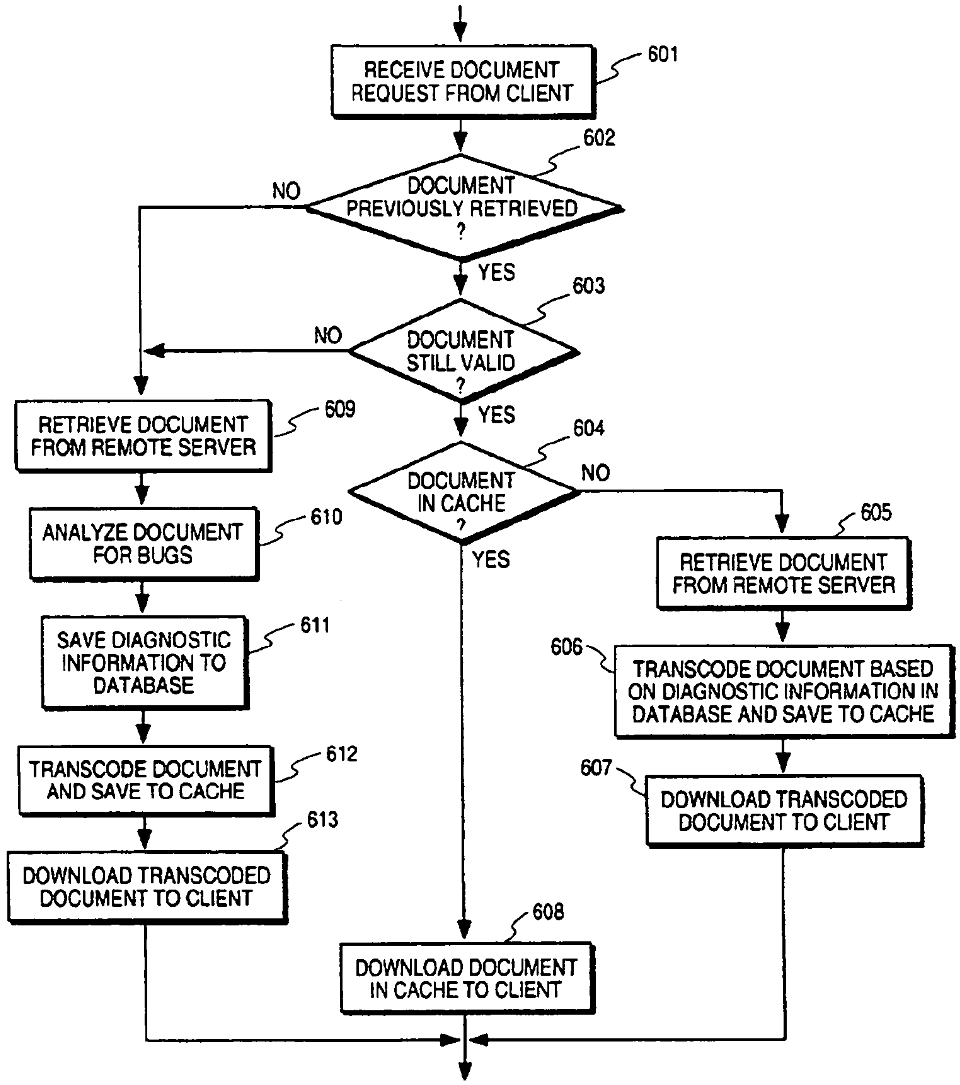

RECEIVE DOCUMENT REQUEST FROM CLIENT
601
DOCUMENT PREVIOUSLY RETRIEVED ?
602
NO
YES
DOCUMENT STILL VALID ?
603
NO
YES
DOCUMENT IN CACHE ?
604
NO
YES
RETRIEVE DOCUMENT FROM REMOTE SERVER
609
ANALYZE DOCUMENT FOR BUGS
610
SAVE DIAGNOSTIC INFORMATION TO DATABASE
611
TRANSCODE DOCUMENT AND SAVE TO CACHE
612
DOWNLOAD TRANSCODED DOCUMENT TO CLIENT
613
RETRIEVE DOCUMENT FROM REMOTE SERVER
605
TRANSCODE DOCUMENT BASED ON DIAGNOSTIC INFORMATION IN DATABASE AND SAVE TO CACHE
606
DOWNLOAD TRANSCODED DOCUMENT TO CLIENT
607
DOWNLOAD DOCUMENT IN CACHE TO CLIENT
608

FIG. 6

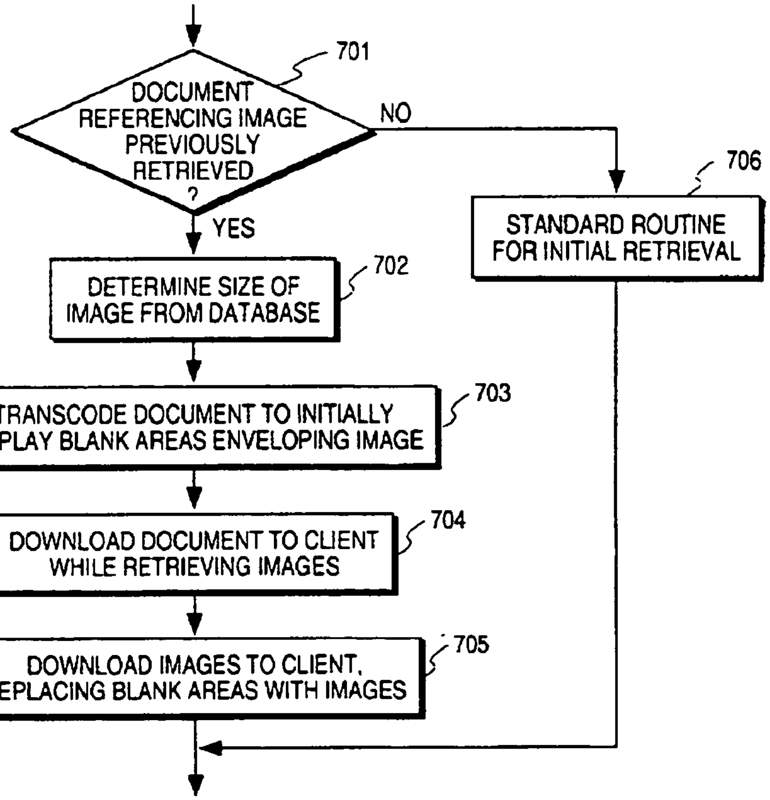

701
DOCUMENT REFERENCING IMAGE PREVIOUSLY RETRIEVED ?
NO
YES
706
STANDARD ROUTINE FOR INITIAL RETRIEVAL
702
DETERMINE SIZE OF IMAGE FROM DATABASE
703
TRANSCODE DOCUMENT TO INITIALLY DISPLAY BLANK AREAS ENVELOPING IMAGE
704
DOWNLOAD DOCUMENT TO CLIENT WHILE RETRIEVING IMAGES
705
DOWNLOAD IMAGES TO CLIENT, REPLACING BLANK AREAS WITH IMAGES

FIG. 7A

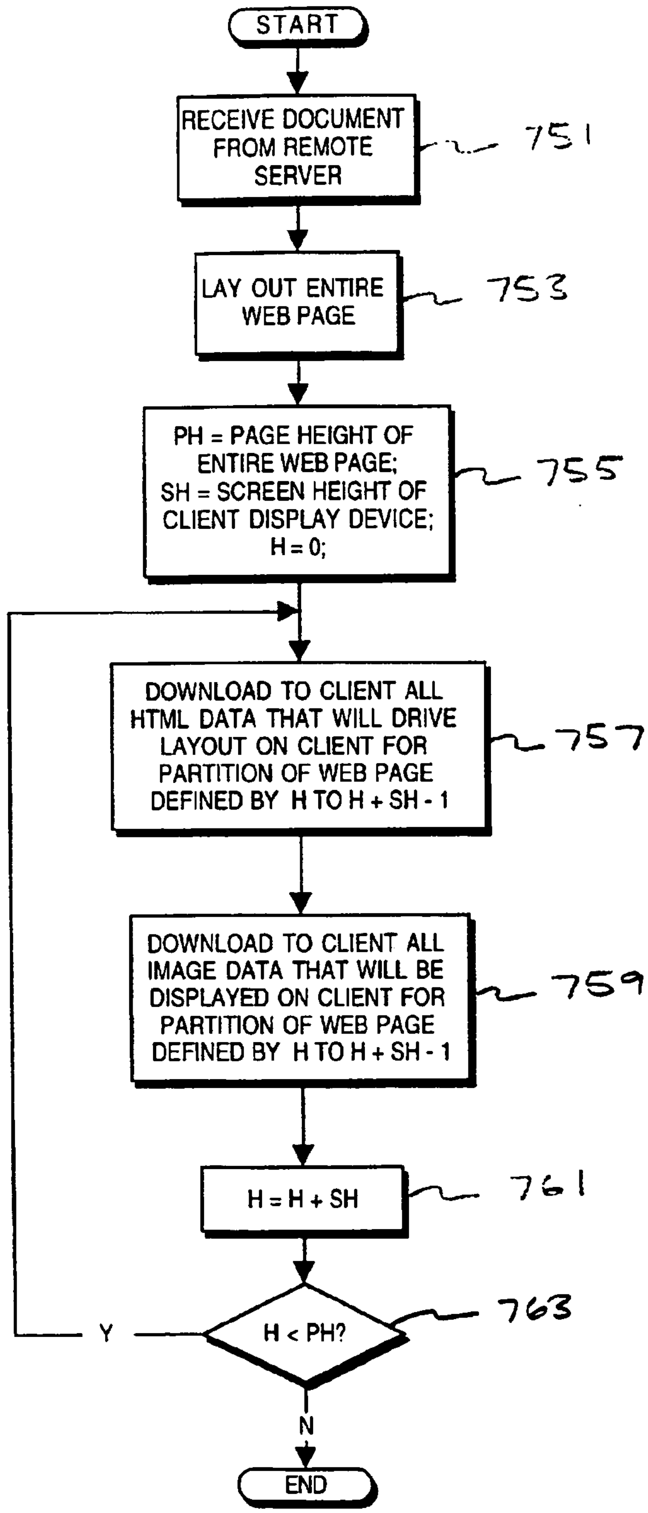

START
RECEIVE DOCUMENT FROM REMOTE SERVER
751
LAY OUT ENTIRE WEB PAGE
753
PH = PAGE HEIGHT OF ENTIRE WEB PAGE; SH = SCREEN HEIGHT OF CLIENT DISPLAY DEVICE; H = 0;
755
DOWNLOAD TO CLIENT ALL HTML DATA THAT WILL DRIVE LAYOUT ON CLIENT FOR PARTITION OF WEB PAGE DEFINED BY H TO H + SH - 1
757
DOWNLOAD TO CLIENT ALL IMAGE DATA THAT WILL BE DISPLAYED ON CLIENT FOR PARTITION OF WEB PAGE DEFINED BY H TO H + SH - 1
759
H = H + SH
761
H < PH?
763
Y
N
END

FIGURE 7B

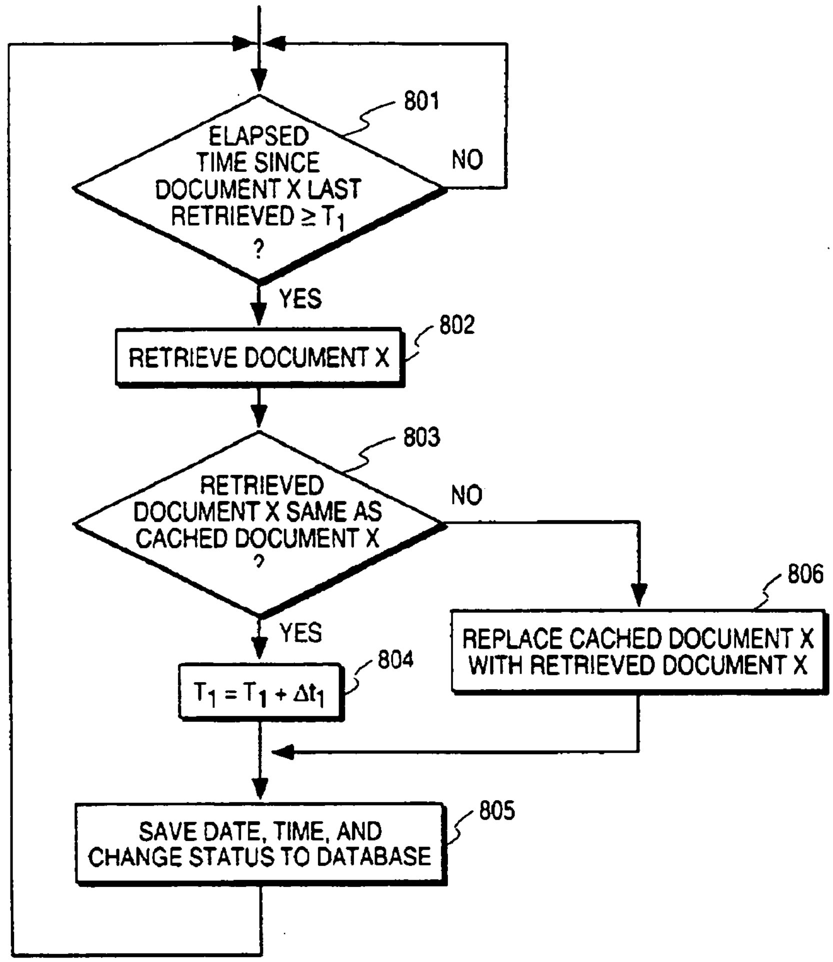

801
ELAPSED TIME SINCE DOCUMENT X LAST RETRIEVED ≥ T_1 ?
NO
YES
802
RETRIEVE DOCUMENT X
803
RETRIEVED DOCUMENT X SAME AS CACHED DOCUMENT X ?
NO
806
REPLACE CACHED DOCUMENT X WITH RETRIEVED DOCUMENT X
YES
804
$T_1 = T_1 + \Delta t_1$
805
SAVE DATE, TIME, AND CHANGE STATUS TO DATABASE

FIG. 8

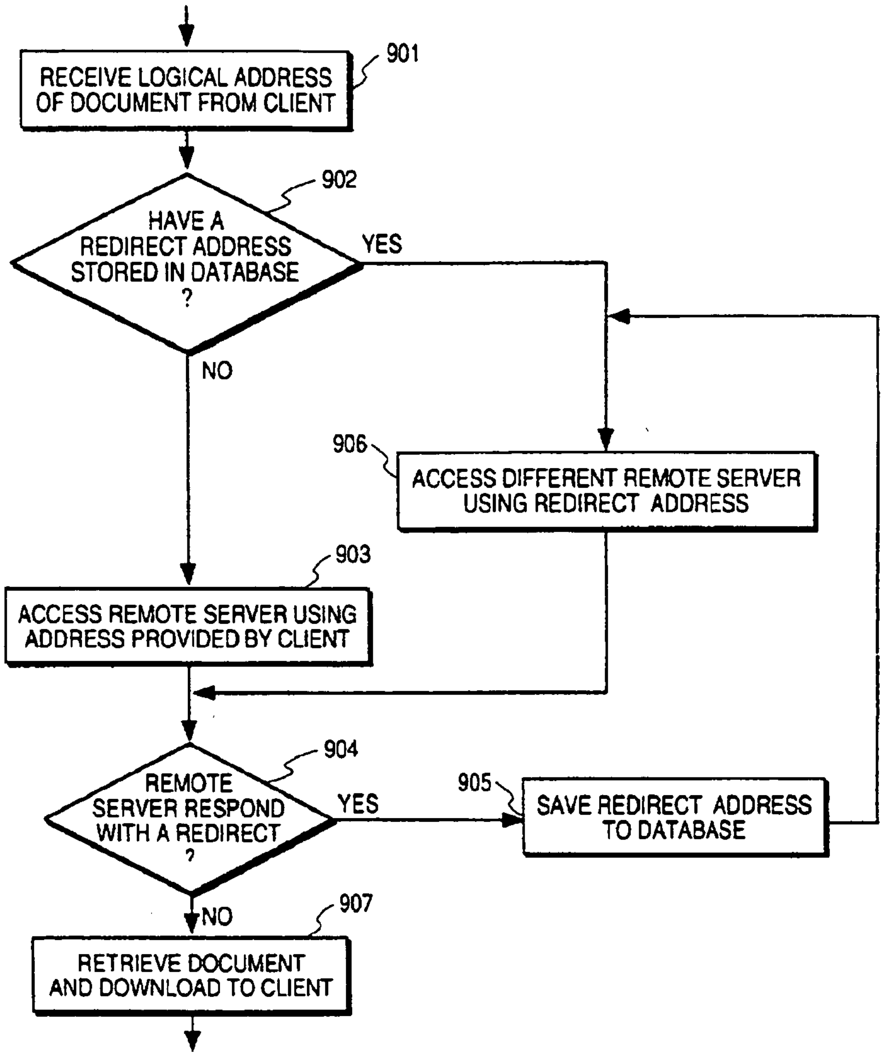

RECEIVE LOGICAL ADDRESS OF DOCUMENT FROM CLIENT
901
HAVE A REDIRECT ADDRESS STORED IN DATABASE ?
902
YES
NO
ACCESS DIFFERENT REMOTE SERVER USING REDIRECT ADDRESS
906
ACCESS REMOTE SERVER USING ADDRESS PROVIDED BY CLIENT
903
REMOTE SERVER RESPOND WITH A REDIRECT ?
904
YES
SAVE REDIRECT ADDRESS TO DATABASE
905
NO
RETRIEVE DOCUMENT AND DOWNLOAD TO CLIENT
907

FIG. 9

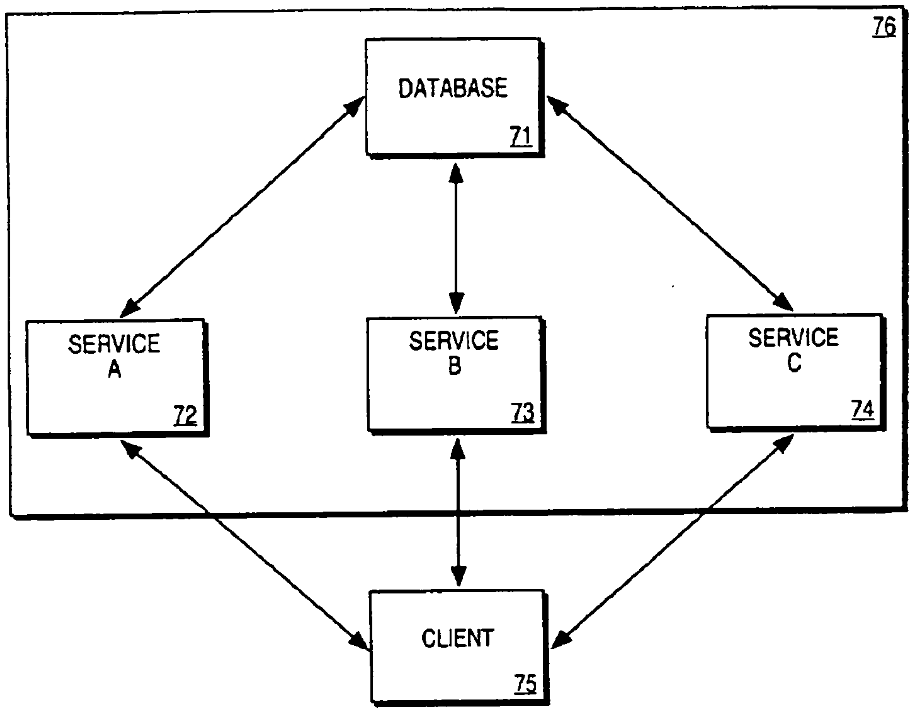


FIG. 10 (PRIOR ART)

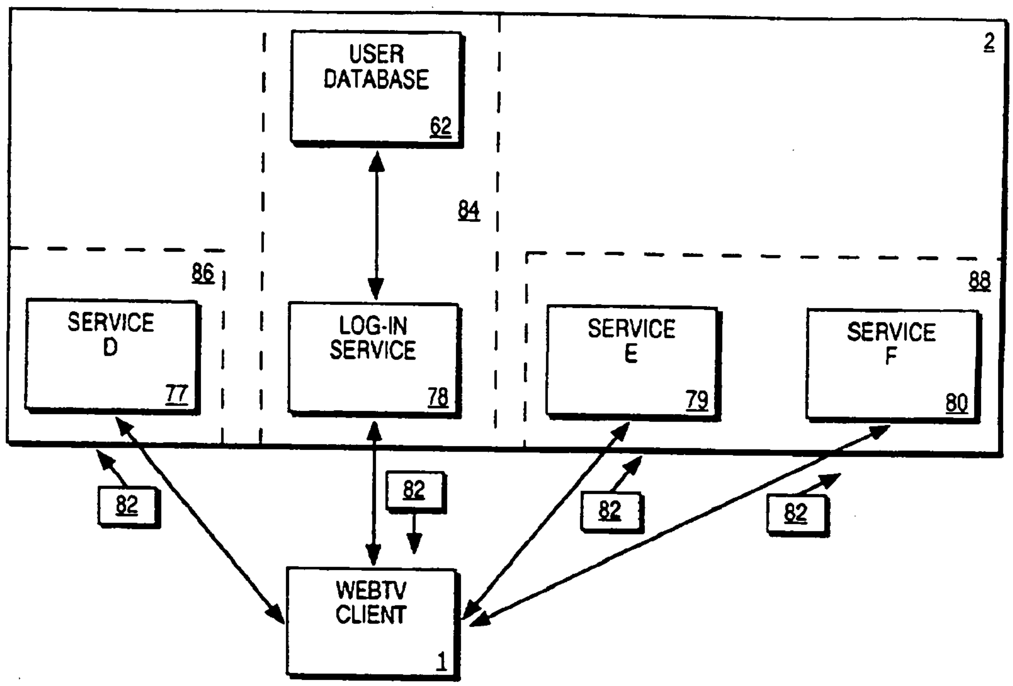

USER
DATABASE
62
2
84
86
88
SERVICE
D
77
LOG-IN
SERVICE
78
SERVICE
E
79
SERVICE
F
80
82
82
82
82
WEBTV
CLIENT
1

FIG. 11

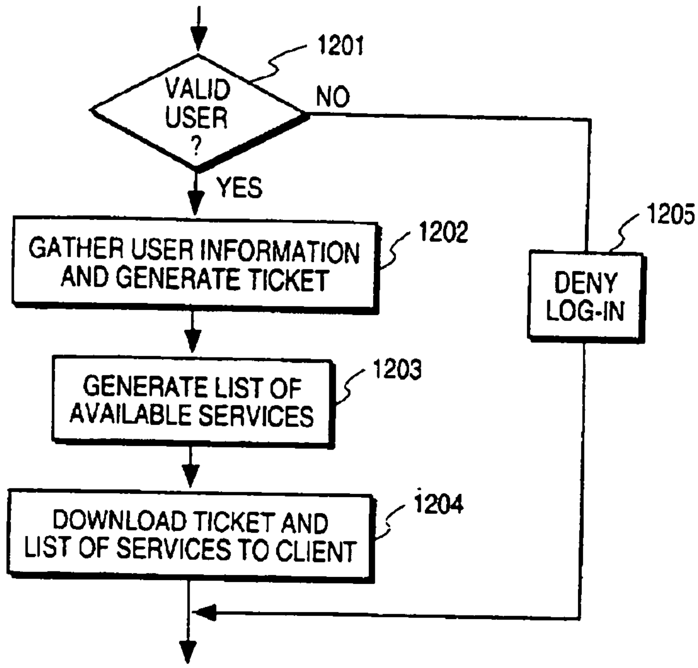

VALID USER ?
1201
NO
YES
GATHER USER INFORMATION AND GENERATE TICKET
1202
DENY LOG-IN
1205
GENERATE LIST OF AVAILABLE SERVICES
1203
DOWNLOAD TICKET AND LIST OF SERVICES TO CLIENT
1204

FIG. 12

METHOD FOR DOWNLOADING A WEB PAGE TO A CLIENT FOR EFFICIENT DISPLAY ON A TELEVISION SCREEN

CROSS-REFERENCES TO RELATED APPLICATIONS

[0001] The present application is related to the following co-pending U.S. patent applications:

[0002] U.S. patent application entitled, "Method and Apparatus for Managing Communications Between a Client and a Server in a Network," having U.S. patent application Ser. No. 08/660,087, and filed on Jun. 3, 1996;

[0003] U.S. patent application entitled, "Web Browser Allowing Navigation Between Hypertext Objects Using Remote Control," having U.S. patent application Ser. No. 08/660,088, and filed on Jun. 3, 1996;

[0004] U.S. patent application entitled, "A Method and Apparatus for Using Network Address Information to Improve the Performance of Network Transactions," having application no. 08/656,923, and filed on Jun. 3, 1996; and

[0005] U.S. patent application entitled, "Method and Apparatus for Providing Proxying and Transcoding of Documents in a Distributed Network," having U.S. patent application Ser. No. 08/656,924, and filed on Jun. 3, 1996; which are signed to the assignee of the present invention.

FIELD OF THE INVENTION

[0006] The present invention pertains to the field of client-server computer networking. More particularly, the present invention relates to a method and apparatus for providing proxying and document transcoding in a server in a computer network.

BACKGROUND OF THE INVENTION

[0007] The number of people using personal computers has increased substantially in recent years, and along with this increase has come an explosion in the use of the Internet. One particular aspect of the Internet which has gained widespread use is the World-Wide Web ("the Web"). The Web is a collection of formatted hypertext pages located on numerous computers around the world that are logically connected by the Internet. Advances in network technology and software providing user interfaces to the Web ("Web browsers") have made the Web accessible to a large segment of the population. However, despite the growth in the development and use of the Web, many people are still unable to take advantage of this important resource.

[0008] Access to the Web has been limited thus far mostly to people who have access to a personal computer. However, many people cannot afford the cost of even a relatively inexpensive personal computer, while others are either unable or unwilling to learn the basic computer skills that are required to access the Web. Furthermore, Web browsers in the prior art generally do not provide the degree of user-friendliness desired by some people, and many computer novices do not have the patience to learn how to use the software. Therefore, it would be desirable to provide an inexpensive means by which a person can access the Web without the use of a personal computer. In particular, it would be desirable for a person to be able to access the Web pages using an ordinary television set and a remote control, so that the person feels more as if he or she is simply changing television channels, rather than utilizing a complex computer network.

[0009] Prior art Web technology also has other significant limitations which can make a person's experience unpleasant when browsing the Web. Web documents are commonly written in HTML (Hypertext Mark-up Language). HTML documents sometimes contain bugs (errors) or have features that are not recognized by certain Web browsers. These bugs or quirks in a document can cause a Web browser to fail. Thus, what is needed is a means for reducing the frequency with which client systems fail due to bugs or quirks in HTML documents.

[0010] Another problem associated with browsing the Web is latency. People commonly experience long, frustrating delays when browsing the Web. It is not unusual for a person to have to wait minutes after selecting a hypertext link for a Web page to be completely downloaded to his computer and displayed on his computer screen. There are many possible causes for latency, such as heavy communications traffic on the Internet and slow response of remote servers. Latency can also be caused by Web pages including images. One reason for this effect is that, when an HTML document references an image, it takes time to retrieve the image itself after the referencing document has been retrieved. Another reason is that, in the prior art, if the referencing document does not specify the size of the image, the client system generally cannot display the Web page until the image itself has been retrieved. Numerous others sources of latency exist with respect to the Web. Therefore, what is needed is a means for reducing such latency, to eliminate some of the frustration which typically has been associated with browsing the Web.

[0011] Security is another concern associated with the Internet. Internet service providers (ISPs) generally maintain certain information about each customer in a database. This information may include information which a customer may not wish to become publicly known, such as social security numbers and credit card numbers. Maintaining the confidentiality of this information in a system that is connected to an expensive publicly-accessible computer network like the Internet can be problematic. Further, the problem can be aggravated by the fact that an ISP often provides numerous different services, each of which has access to this database. Allowing access to the database by many different entities creates many opportunities for security breaches to occur. Therefore, what is needed is a way to improve the security of confidential customer information in a server system coupled to the Internet.

SUMMARY OF THE INVENTION

[0012] An improved method for providing a document to a client coupled to a server is disclosed. In the method, a document is provided to a proxying server. The document includes image data and non-image data that causes the client to generate a display. The document is partitioned into a plurality of partitions. The data of the first partition is downloaded to the client after the data of the first partition is downloaded, the data of the next partition is downloaded to the client. These steps are repeated until each one of the plurality of partitions has been downloaded to the client.

Other features of the present invention will be apparent from the accompanying drawings and from the detailed description which follows method is described of providing a document to a client coupled to a server.

BRIEF DESCRIPTION OF THE DRAWINGS

[0013] The present invention is illustrated by way of example and not limitation in the figures of the accompanying drawings, in which like references indicate similar elements and in which:

[0014] FIG. 1 illustrates several clients connected to a proxying server in a network.

[0015] FIG. 2 illustrates a client according to the present invention.

[0016] FIG. 3 is a block diagram of a server according to the present invention.

[0017] FIG. 4A illustrates a server including a proxy cache and a transcoder.

[0018] FIG. 4B illustrates databases used in a server according to the present invention.

[0019] FIG. 5 is a flow diagram illustrating a routine for transcoding a document retrieved from a remote server using data stored in a persistent database.

[0020] FIG. 6 is a flow diagram illustrating a routine for transcoding an HTML document for purposes of eliminating bugs or undesirable features.

[0021] FIG. 7A is a flow diagram illustrating a routine for reducing latency when downloading a document referencing an image to a client.

[0022] FIG. 7B is a flow diagram illustrating a routine for efficiently downloading a document to a client for efficient display of a Web page on a television screen.

[0023] FIG. 8 is a flow diagram illustrating a routine for updating documents stored in the proxy cache using data stored in a persistent database.

[0024] FIG. 9 is a flow diagram illustrating a routine used by a server for retrieving documents from another remote server.

[0025] FIG. 10 is a block diagram of a prior art server system showing a relationship between various services and a database.

[0026] FIG. 11 is a block diagram of a server system according to the present invention showing a relationship between various services and a user database.

[0027] FIG. 12 is a flow diagram illustrating a routine used by a server for regulating access to various services provided by the server.

DETAILED DESCRIPTION

[0028] A method and apparatus are described for providing proxying and transcoding of documents in a network. In the following description, for purposes of explanation, numerous specific details are set forth in order to provide a thorough understanding of the present invention. It will be evident, however, to one skilled in the art that the present invention may be practiced without these specific details. In other instances, well-known structures and devices are shown in block diagram form in order to avoid unnecessarily obscuring the present invention.

[0029] The present invention includes various steps, which will be described below. The steps can be embodied in machine-executable instructions, which can be used to cause a general-purpose or special-purpose processor programmed with the instructions to perform the steps. Alternatively, the steps of the present invention might be performed by specific hardware components that contain hardwired logic for performing the steps, or by any combination of programmed computer components and custom hardware components.

I. System Overview

[0030] The present invention is included in a system, known as WebTV™, for providing a user with access to the Internet. A user of a WebTV™ client generally accesses a WebTV™ server via a direct-dial telephone (POTS, for "plain old telephone service"), ISDN (Integrated Services Digital Network), or other similar connection, in order to browse the Web, send and receive electronic mail (e-mail), and use various other WebTV™ network services. The WebTV™ network services are provided by WebTV™ servers using software residing within the WebTV™ servers in conjunction with software residing within a WebTV™ client.

[0031] FIG. 1 illustrates a basic configuration of the WebTV™ network according to one embodiment. A number of WebTV™ clients 1 are coupled to a modem pool 2 via direct-dial, bidirectional data connections 29, which may be telephone (POTS, i.e., "plain old telephone service"), ISDN (Integrated Services Digital Network), or any other similar type of connection. The modem pool 2 is coupled typically through a router, such as that conventionally known in the art, to a number of remote servers 4 via a conventional network infrastructure 3, such as the Internet. The WebTV™ system also includes a WebTV™ server 5, which specifically supports the WebTV™ clients 1. The WebTV™ clients 1 each have a connection to the WebTV™ server 5 either directly or through the modem pool 2 and the Internet 3. Note that the modem pool 2 is a conventional modem pool, such as those found today throughout the world providing access to the Internet and private networks.

[0032] Note that in this description, in order to facilitate explanation the WebTV™ server 5 is generally discussed as if it were a single device, and functions provided by the WebTV™ services are generally discussed as being performed by such single device. However, the WebTV™ server 5 may actually comprise multiple physical and logical devices connected in a distributed architecture, and the various functions discussed below which are provided by the WebTV™ services may actually be distributed among multiple WebTV™ server devices.

II. Client System

[0033] FIG. 2 illustrates a WebTV™ client 1. The WebTV™ client 1 includes an electronics unit 10 (hereinafter referred to as "the WebTV™ box 10"), an ordinary television set 12, and a remote control 11. In an alternative embodiment of the present invention, the WebTV™ box 10 is built into the television set 12 as an integral unit. The

WebTV™ box 10 includes hardware and software for providing the user with a graphical user interface, by which the user can access the WebTV™ network services, browse the Web, send e-mail, and otherwise access the Internet.

[0034] The WebTV™ client 1 uses the television set 12 as a display device. The WebTVT™ box 10 is coupled to the television set 12 by a video link 6. The video link 6 is an RF (radio frequency), S-video, composite video, or other equivalent form of video link. In the preferred embodiment, the client 1 includes both a standard modem and an ISDN modem, such that the communication link 29 between the Web™ box 10 and the server 5 can be either a telephone (POTS) connection 29*a* or an ISDN connection 29*b*. The WebTV™ box 10 receives power through a power line 7.

[0035] Remote control 11 is operated by the user in order to control the WebTV™ client 1 in browsing the Web, sending e-mail, and performing other Internet-related functions. The WebTV™ box 10 receives commands from remote control 11 via an infrared (IR) communication link. In alternative embodiments, the link between the remote control 11 and the WebTV™ box 10 may be RF or any equivalent mode of transmission.

III. Server System

[0036] The WebTV™ server 5 generally includes one or more computer systems generally having the architecture illustrated in FIG. 3. It should be noted that the illustrated architecture is only exemplary; the present invention is not constrained to this particular architecture. The illustrated architecture includes a central processing unit (CPU) 50, random access memory (RAM) 51, read-only memory (ROM) 52, a mass storage device 53, a modem 54, a network interface card (NIC) 55, and various other input/output (I/O) devices 56. Mass storage device 53 includes a magnetic, optical, or other equivalent storage medium. I/O devices 56 may include any or all of devices such as a display monitor, keyboard, cursor control device, etc.. Modem 54 is used to communicate data to and from remote servers 4 via the Internet.

[0037] As noted above, the WebTV™ server 5 may actually comprise multiple physical and logical devices connected in a distributed architecture. Accordingly, NIC 55 is used to provide data communication with other devices that are part of the WebTV™ services. Modem 54 may also be used to communicate with other devices that are part of the WebTV™ services and which are not located in close geographic proximity to the illustrated device.

[0038] According to the present invention, the Web™ server 5 acts as a proxy in providing the WebTV™ client 1 with access to the Web and other WebTV™ services. More specifically, WebTV™ server 5 functions as a "caching proxy."FIG. 4A illustrates the caching feature of the WebTV™ server 5. In FIG. 4A, the WebTV™ server 5 is functionally located between the WebTV™ client 1 and the Internet infrastructure 3. The WebTV™ server 5 includes a proxy cache 65 which is functionally coupled to the WebTV™ client 1. The proxy cache 65 is used for temporary storage of Web documents, images, and other information which is used by frequently either the WebTV™ client 1 or the WebTV™ server 5.

[0039] A document transcoder 66 is functionally coupled between the proxy cache 65 and the Internet infrastructure 3. The document transcoder 66 includes software which is used to automatically revise the code of Web documents retrieved from the remote servers 4, for purposes which are described below.

[0040] The WebTV™ service provides a document database 61 and a user database 62, as illustrated in FIG. 4B. The user database 62 contains information that is used to control certain features relating to access privileges and capabilities of the user of the client 1. This information is used to regulate initial access to the WebTV™ service, as well as to regulate access to the individual services provided by the WebTV™ system, as will be described below. The document database 61 is a persistent database which stores certain diagnostic and historical information about each document and image retrieved by the server 5, as is now described.

A. Document Database

[0041] The basic purpose of the document database 61 is that, after a document has once been retrieved by the server 5, the stored information can be used by the server 5 to speed up processing and downloading of that document in response to all future requests for that document. In addition, the transcoding functions and various other functions of the WebTV™ service are facilitated by making use of the information stored in the document database 61, as will be described below.

[0042] Referring now to FIG. 5, the server 5 initially receives a document request from a client 1 (step 501). The document request will generally result from the user of the client 1 activating a hypertext anchor (link) on a Web page. The act of activating a hypertext anchor may consist of clicking on underlined text in a displayed Web page using a mouse, for example. The document request will typically (but not always) include the URL (Uniform Resource Locator) or other address of the selected anchor. Upon receiving the document request, the server 5 optionally accesses the document database 62 to retrieve stored information relating to the requested document (step 502). It should be noted that the document database 62 is not necessarily accessed in every case. The information retrieved from the document database 62 is used by the server 5 for determining, among other things, how long a requested document has been cached and/or whether the document is still valid. The criteria for determining validity of the stored document are discussed below. The server 5 retrieves the document from the cache 65 if the stored document is valid; otherwise, the server 5 retrieves the document from the appropriate remote server 4 (step 503). The server 5 automatically transcodes the document as necessary based on the information stored in the document database 61 (step 503). The transcoding functions are discussed further below.

[0043] The document database 61 includes certain historical and diagnostic information for every Web page that is accessed at any time by a WebTV™ client 1. As is well known, a Web page may correspond to a document written in a language such as HTML (Hypertext Mark-Up Language), VRML (Virtual Reality Modelling Language), or another suitable language. Alternatively, a Web page may represent an image, or a document which references one or more images. According to the present invention, once a document or image is retrieved by the WebTV™ server 5

from a remote server 4 for the first time, detailed information on this document or image is stored permanently in the document database 61. More specifically, for every Web page that is retrieved from a remote server 4, any or all of the following data are stored in the document database 61:

[0044] (1) information identifying bugs (errors) or quirks in the Web page, or undesirable effects caused when the Web page is displayed by a client 1;

[0045] (2) relevant bug-finding algorithms;

[0046] (3) the date and time the Web page was last retrieved;

[0047] (4) the date and time the Web page was most recently altered by the author;

[0048] (5) a checksum for determining whether the Web page has been altered;

[0049] (6) the size of the Web page (in terms of memory);

[0050] (7) the type of Web page (e.g., HTML document, image, etc.);

[0051] (8) a list of hypertext anchors (links) in the Web page and corresponding URLs;

[0052] (9) a list of the most popular anchors based on the number of "hits" (requests from a client 1);

[0053] (10) a list of related Web pages which can be prefetched

[0054] (11) whether the Web page has been redirected to another remote server 4;

[0055] (12) a redirect address (if appropriate);

[0056] (13) whether the redirect (if any) is temporary or permanent, and if permanent, the duration of the redirect;

[0057] (14) if the Web page is an image, the size of the image in terms of both physical dimensions and memory space;

[0058] (15) the sizes of in-line images (images displayed in text) referenced by the document defining the Web page;

[0059] (16) the size of the largest image referenced by the document;

[0060] (17) information identifying any image maps in the Web page;

[0061] (18) whether to resize any images corresponding to the Web page;

[0062] (19) an indication of any forms or tables in the Web page;

[0063] (20) any unknown protocols;

[0064] (21) any links to "dead" Web pages (i.e., pages which are no longer active);

[0065] (22) the latency and throughput of the remote server 4 on which the Web page is located;

[0066] (23) the character set of the document;

[0067] (24) the vendor of the remote server 4 on which the Web page is located;

[0068] (25) the geographic location of the remote server 4 on which the Web page is located;

[0069] (26) the number of other Web pages which reference the subject Web page;

[0070] (27) the compression algorithm used by the image or document;

[0071] (28) the compression algorithm chosen by the transcoder;

[0072] (29) a value indicating the popularity of the Web page based on the number of hits by clients; and

[0073] (30) a value indicating the popularity of other Web pages which reference the subject Web page.

B. Transcoding

[0074] As mentioned above, the WebTV™ services provide a transcoder 66, which is used to rewrite certain portions of the code in an HTML document for various purposes. These purposes include: (1) correcting bugs in documents; (2) correcting undesirable effects which occur when a document is displayed by the client 1; (3) improving the efficiency of transmission of documents from the server 5 to the client 1; (4) matching hardware decompression technology within the client 1; (5) resizing images to fit on the television set 12; (6) converting documents into other formats to provide compatibility; (7) reducing latency experienced by a client 1 when displaying a Web page with in-line images (images displayed in text); and, (8) altering documents to fit into smaller memory spaces.

[0075] There are three transcoding modes used by the transcoder 66: (1) streaming, (2) buffered, and (3) deferred. Streaming transcoding refers to the transcoding of documents on a line-by-line basis as they are retrieved from a remote server 4 and downloaded to the client 1 (i.e., transcoding "on the fly"). Some documents, however, must first be buffered in the WebTV™ server 5 before transcoding and downloading them to the client 1. A document may need to be buffered before transmitting it to the client 1 if the type of changes to be made can only be made after the entire document has been retrieved from the remote server 4. Because the process of retrieving and downloading a document to the client 1 increases latency and decreases throughput, it is not desirable to buffer all documents. Therefore, the transcoder 66 accesses and uses information in the document database 61 relating to the requested document to first determine whether a requested document must be buffered for purposes of transcoding, before the document is retrieved from the remote server 4.

[0076] In the deferred mode, transcoding is deferred until after a requested document has been downloaded to a client 1. The deferred mode therefore reduces latency experienced by the client 1 in receiving the document. Transcoding may be performed immediately after downloading or any time thereafter. For example, it may be convenient to perform transcoding during periods of low usage of WebTV™ services, such as at night. This mode is useful for certain types of transcoding which are not mandatory.

1. Transcoding for Bugs and Quirks

[0077] One characteristic of some prior art Web browsers is that they may experience failures ("crashes") because of bugs or unexpected features ("quirks") that are present in a Web document. Alternatively, quirks in a document may

cause an undesirable result, even though the client does not crash. Therefore, the transcoding feature of the present invention provides a means for correcting certain bugs and quirks in a Web document. To be corrected by the transcoder **66**, bugs and quirks must be identifiable by software running on the server **5**. Consequently, the transcoder **66** will generally only correct conditions which have been previously discovered, such as those discovered during testing or reported by users. Once a bug or quirk is discovered, however, algorithms are added to the transcoder **66** to both detect the bug or quirk in the future in any Web document and to automatically correct it.

[0078] There are countless possibilities of bugs or quirks which might be encountered in a Web document. Therefore, no attempt will be made herein to provide an exhaustive list. Nonetheless, some examples may be useful at this point. Consider, for example, an HTML document that is downloaded from a remote server **4** and which contains a table having a width specified in the document as "0." This condition might cause a failure if the client were to attempt to display the document as written. This situation therefore, can be detected and corrected by the transcoder **66**. Another example is a quirk in the document which causes quotations to be terminated with too many quotation marks. Once the quirk is first detected and an algorithm is written to recognize it, the transcoder **66** can automatically correct the quirk in any document.

[0079] If a given Web document has previously been retrieved by the server **5**, there will be information regarding that document available in the document database **61** as described above. The information regarding this document will include whether or not the document included any bugs or quirks that required transcoding when the document was previously retrieved. The transcoder **66** utilizes this information to determine whether (1) the document is free of bugs and quirks, (2) the document has bugs or quirks which can be remedied by transcoding on the fly, or (3) the document has bugs or quirks which cannot be corrected on the fly (i.e., buffering is required).

[0080] **FIG. 6** illustrates a routine for transcoding a Web document for purposes of eliminating bugs and quirks. Initially, the server **5** receives a document request from the client **1** (step **601**). Next, the document database **61** is accessed to determine whether or not the requested document has been previously retrieved (step **602**). If the document has not been previously retrieved, then the server **5** retrieves the document from the remote server **4** (step **609**). Next, the retrieved document is analyzed for the presence of bugs or unusual conditions (step **610**). Various diagnostic information is then stored in the document database **61** as a result of the analysis to note any bugs or quirks that were found (step **611**). If any bugs or quirks were found which can be corrected by the transcoder **66**, the document is then transcoded and saved to the proxy cache **65** (step **612**). The transcoded document is then downloaded to the client **1** (step **613**). It should be noted that transcoding can be deferred until after the document has been downloaded, as described above; hence, the sequence of **FIG. 6** is illustrative only.

[0081] If (in step **602**) the requested document had been previously retrieved, then it is determined whether the requested document is still valid (step **603**) and whether the document is present in the proxy cache **65** (step **604**). If the document is no longer valid, then the document is retrieved from the remote server **4**, analyzed for bugs and quirks, transcoded as required, and then downloaded to the client **1** as described above (steps **610-613**, step **607**). Methods for determining validity of a document are discussed below. If the document is still valid (step **603**) and the document is present in the cache **65**, the document is downloaded to the client **1** in its current form (as it is stored in the cache), since it has already been transcoded (step **608**).

[0082] The document, however, may be valid but not present in the cache. This may be the case, for example, if the document has not been requested recently and the cache **65** has become too full to retain the requested document. In that case, the document is retrieved again from the remote server **4** (step **605**) and then transcoded on the basis of the previously-acquired diagnostic information stored within the database **61** for that document. The document is then saved to the cache **65** (step **606**). Note that because the document is still valid, it is assumed that the diagnostic information stored in the document database **61** for that document is still valid and that the transcoding can be performed on the basis of that information. Accordingly, once the document is transcoded, the transcoded document is downloaded to the client **1** (step **607**). Again, note that transcoding can be deferred until after the document has been downloaded in some cases.

[0083] The validity of the requested document can be determined based on various different criteria. For example, some HTML documents specify a date on which the document was created, a length of time for which the document will be valid, or both. The validity determination can be based upon such information. For example, a document which specifies only the date of creation can be automatically deemed invalid after a predetermined period of time has passed.

[0084] Alternatively, validity can be based upon the popularity of the requested document. "Popularity" can be quantified based upon the number of hits for that document, which is tracked in the document database **61**. For example, it might be prudent to simply assign a relatively short period of validity to a document which is very popular and a longer period of validity to a document which is less popular.

[0085] Another alternative basis for the validity of a document is the observed rate of change of the document. Again, data in the persistent document database **61** can be used. That is, because the document database **61** stores the date and time on which the document was last observed to change, the server **5** can approximate how often the document actually changes. A document or image which is observed to change frequently (e.g., a weather map or a news page) can be assigned a relatively short period of validity. It will be recognized that numerous other ways of determining validity are possible.

2. Transcoding to Reduce Latency

[0086] Another purpose for transcoding is to allow documents requested by a client **1** to be displayed by the client **1** more rapidly. Many HTML documents contain references to "in-line" images, or images that will be displayed in text in a Web page. The normal process used in the prior art to display a Web page having in-line images is that the HTML

document referencing the image is first downloaded to the client, followed by the client's requesting the referenced image. The referenced image is then retrieved from the remote server on which it is located and downloaded to the client. One problem associated with the prior art, however, is that the speed with which a complete Web page can be displayed to the user is often limited by the time it takes to retrieve in-line images. One reason for this is that it simply takes time to retrieve the image itself after the referencing document has been retrieved. Another reason is that, in the prior art, if the referencing document does not specify the size of the image, the Web page generally cannot be displayed until the image itself has been retrieved. The present invention overcomes these limitations.

[0087] According to the present invention, information stored in the document database **61** regarding the in-line images is used to transcode the referencing document in order to reduce latency in displaying the Web page. Once any document which references an in-line image is initially retrieved by the server **5**, the fact that the document references an in-line image is stored in the document database **61**. In addition, the size of the image is determined, either from the document (if specified) or from the image itself, and then stored in the document database **61**. Consequently, for documents which do not specify the size of their in-line images, the size information stored in the database **61** is then used the next time the document is requested in order to reduce latency in downloading and displaying the Web page.

[0088] Refer now to **FIG. 7A**, which illustrates a routine for reducing latency when downloading a document referencing an image to a client **1**. Assume that a client **1** sends a request to the server **5** for an HTML document containing a reference to an in-line image. Assume further that the size of the image is not specified in the document itself. Initially, the server **5** determines whether that document has been previously retrieved (step **701**). If not, the standard initial retrieval and transcoding procedure is followed (step **706**), as described in connection with **FIG. 6**. If, however, the document has been previously retrieved, then the transcoder **66** accesses the size information stored in the document database **61** for the in-line image (step **702**). Based on this size information, the HTML document is transcoded such that, when the Web page is initially displayed by the client **1**, the area in which the image belongs is replaced by a blank region enveloping the shape of the image. Thus, any in-line image referenced by a document is displayed initially as a blank region. Consequently, the client **1** can immediately display the Web page corresponding to the HTML document even before the referenced image has been retrieved or downloaded (i.e., even before the size of the image is known to the client **1**).

[0089] As the transcoded HTML document is downloaded to the client, the image is retrieved from the appropriate remote server **4** (step **704**). Once the image is retrieved from the remote server **4** and downloaded to the client **1**, the client **1** replaces the blank area in the Web page with the actual image (step **705**).

3. Transcoding to Display Web Pages on a Television

[0090] As noted above, the client **1** utilizes an ordinary television set **12** as a display device. However, images in Web pages are generally formatted for display on a computer monitor, not a television set. Consequently, the transcoding function of the present invention is used to resize images for display on the television set **12**. This includes rescaling images as necessary to avoid truncation when displayed on the television set **12**.

[0091] It should be noted that prior art Web browsers which operate on computer monitors typically use resizable windows. Hence, the size of the visible region varies from client to client. However, because the web browser used by the WebTV™ client **1** is specifically designed for display on a television set, the present invention allows documents and images to be formatted when they are cached.

[0092] As mentioned previously, prior art servers generally download all of the HTML data, or non-image data, of a Web page to a client and then subsequently download the image data to the client when displaying a Web page, without consideration of the viewable display area of the screen. A trade-off with this prior art approach is that it optimizes total Web page throughput at the expense of the time required to update the viewable display area of the screen. Although this problem may not be as noticeable on computers with large monitors having large display areas, this problem is more noticeable on displays with relatively smaller viewer areas, such as for example a television set **12** coupled to WebTV™ client **1**.

[0093] In another aspect of the present invention, server **5** addresses this issue by taking into consideration the viewable display area of the screen and thus changing the order in which Web based information is downloaded from server **5** to WebTV™ client **1** for the efficient display of a Web page on television set **12**. In this embodiment, server **5** reorders the Web page information such that all of the non-image data and image data that appears within the viewable display area of the screen of television set **12** is downloaded before the non-image data and image data of the Web page that is outside the viewable display area of the screen of television set **12** is downloaded. As a result, the overall time required by WebTV™ client **1** to fully generate and display just the portion of the Web page within the viewable display area of television set **12** is reduced.

[0094] **FIG. 7B** is a flow diagram illustrating the steps performed to efficiently download Web page data from server **5** to a WebTV™ client **1** for efficient display within the viewable display area of a television screen in accordance with the teachings of the present invention. Assume that WebTV™ client **1** sends a request to server **5** for a document that is used to generate a Web page. Assume further that the document contains non-image data, such as for example HTML data, and image data. Assume also that the overall size of the Web page is such that the entire Web page cannot be displayed on a single screen of television set **12**.

[0095] Initially, server **5** receives the document that defines the Web page from remote server **4** (step **751**). After the document has been received from remote server **4**, server **5** lays out the entire Web page defined by the received document using well known techniques (step **753**). After the entire Web page has been laid out in server **5**, the Web page may be separated or partitioned into a plurality of viewable portions or partitions, such that each partition corresponds to one screen of Web page information viewable on the tele-

vision screen. In one embodiment, one television screen of Web page information corresponds to the viewable display height of the television screen.

[0096] In one embodiment, the page height of the entire Web page defined by the document is assumed to equal PH and the display or screen height of television set 12 coupled to WebTV™ client 1 is presumed to equal SH (step 755). A variable H is also initialized to a value of 0 (step 755). Next, all of the non-image data that drives the layout on WebTV™ client 1 within the viewable display area on television set 12 is downloaded to WebTV™ client 1. In one embodiment, this non-image data that is transmitted or downloaded includes all of the HTML data that drives the layout on WebTV™ client 1 within the partition of the Web page defined by the region H to H+SH−1 (step 757). After all the non-image data that drives the layout within the viewable display area on the screen of television set 12 has been downloaded, server 5 then downloads to WebTV™ client 1 all of the image data that is displayed within the viewable display area of the screen of television set 12. The image data downloaded from server 5 to WebTV™ client 1 is defined to be the image data displayed by the client within the partition of the Web page defined by H to H+SH−1 (step 759).

[0097]. After the non-image data and image data have been downloaded as described from server 5 to WebTV™ client 1 for a first television screen of Web page information, server 5 then sequentially repeats the steps of downloading the non-image data and the image data for each of the remaining viewable television screens of the Web page information until all of the television screens of the Web page have been downloaded from server 5 to WebTV™ client 1. In one embodiment, each remaining television screen of the Web page information is not downloaded until all of the non-image data and image data of a previous television screen Web page have been fully transmitted or downloaded.

[0098] Referring back to FIG. 7B, the remaining partitions, or television screens of the Web page information, are downloaded from server 5 to WebTV™ client 1 according to the following remaining steps. After step 759, H is incremented by SH (step 761). If H is less than PH, then processing is looped back to step 757 (step 763). If H is not less than PH, then all of the partitions, or television screens of the Web page, have been downloaded from server 5 to WebTV™ client 1. By reording the Web page information as described, each television screen of a Web page information is efficiently downloaded from server 5 to WebTV™ client 1 and may therefore be fully generated and displayed on television set 12 in a reduced amount of time.

[0099] 4. Transcoding for Transmission Efficiency

[0100] Documents retrieved by the server 5 are also transcoded to improve transmission efficiency. In particular, documents can be transcoded in order to reduce high frequency components in order to reduce interlace flicker when they are displayed on a television set. Various methods for coding software or hardware to reduce perceptual interlace flicker are described in co-pending U.S. patent application Ser. No. ______, filed on ______.

[0101] Documents can also be transcoded in order to lower the resolution of the displayed Web page. Reducing the resolution is desirable, because images formatted for computer systems will generally have a higher resolution than the NTSC (National Television Standards Committee) video format used by conventional television sets. Since the NTSC video does not have the bandwidth to reproduce the resolution of computer-formatted images, the bandwidth consumed in transmitting images to the client 1 at such a high resolution would be wasted.

5. Other Uses for Transcoding

[0102] Transcoding is also used by the present invention to recode a document using new formats into older, compatible formats. Images are often displayed in the JPEG (Joint Picture Experts Group) format or the GIF image format. JPEG often consumes less bandwidth than GIF, however. Consequently, images which are retrieved in GIF format are sometimes transcoded into JPEG format. Methods for generally converting images between GIF and JPEG formats are well known.

[0103] Other uses for transcoding include transcoding audio files. For example, audio may be transcoded into different formats in order to achieve a desired balance between memory usage, sound quality, and data transfer rate. In addition, audio may be transcoded from a file format (e.g., an ".AU" file) to a streaming format (e.g., MPEG 1 audio). Yet another use of audio transcoding is the transcoding of MIDI (Musical Instrument Digital Interface) data to streaming variants of MIDI.

[0104] Additionally, documents or images requiring a large amount of memory (e.g., long lists) can be transcoded in order to consume less memory space in the client 1. This may involve, for example, separating a large document or image into multiple sections. For example, the server 5 can insert tags at appropriate locations in the original document so that the document appears to the client 1 as multiple Web pages. Hence, while viewing a given page representing a portion of the original document, the user can view the next page (i.e., the next portion of the original document) by activating a button on the screen as if it were an ordinary hypertext anchor.

C. Proxying

[0105] As noted above, the server 5 functions as a proxy on behalf of the client 1 for purposes of accessing the Web. The document database 61 is used in various ways to facilitate this proxy role, as will now be described.

1. Updating Cached Documents

[0106] It is desirable to store frequently-requested HTML documents and images in the proxy cache 65 to further reduce latency in providing Web pages to the client 1. However, because some documents and images change over time, documents in the cache 65 will not be valid indefinitely, as mentioned above. A weather map or a news-related Web page, for example, are likely to be updated quite frequently. Consequently, it is desirable for the server 5 to have the ability to estimate the frequency with which documents change, in order to determine how long a document can safely remain within the proxy cache 65 without being updated.

[0107] The persistent database 65 is used to store the date and time of the last several fetches of each document and

image retrieved from a remote server 4, along with an indication of any changes that were detected, if any. A document or image which has been stored in the cache 65 is then retrieved on a periodic basis to determine if it has been changed. Change status information indicating whether the document has changed since the previous fetch is then stored in the document database 65. If no changes are detected, then the time interval between fetches of this document is increased. If the document has changed, the time interval is maintained or decreased. As a result, items in the cache 65 which change frequently will be automatically updated at frequent intervals, whereas documents which do not change often will be replaced in the cache less frequently.

[0108] **FIG. 8** illustrates a routine for updating documents stored in the proxy cache **65** using data stored in the document database **61**. Assume a document X has been stored in the proxy cache **65**. Document X remains in the cache **65** until a predetermined update period Ti expires (step **801**). Upon the expiration of the update period T_1, the document X is again retrieved from the appropriate remote server **4** (step **802**). The newly-retrieved document X is then compared to the cached version of document X (step **803**). If the document has changed, then the cached version of document X is replaced with the newly-retrieved version of document X (step **806**). If not, then the update period T_1 is increased according to a predetermined time increment Dt_1 (step **804**). In any case, the date and time and the change status of document X is saved to the document database **61** (step **805**).

Document and Image Prefetching

[0109] The document database **61** is also used by the server **5** to store prefetching information relating to documents and images. In particular, the database stores, for each document that has been retrieved, a list of images referenced by the document, if any, and their locations. Consequently, the next time a document is requested by a client **1**, the images can be immediately retrieved by the server **5** (from the cache **65**, if available, or from the remote server **4**), even before the client **1** requests them. This procedure improves the speed with which requested Web pages are downloaded to the client.

[0110] The document database **61** is also used to facilitate a process referred to as "server-advised client prefetching." Server-advised client prefetching allows the server **5** to inform the client **1** of documents or images which are popular to allow the client **1** to perform the prefetching. In particular, for any given document, a list is maintained in the server **5** of the most popular hypertext anchors in that document (i.e., those which have previously received a large number of hits). When that document is requested by the client **1**, the server **5** provides the client **1** with an indication of these popular links.

3. Redirects

[0111] Web pages are sometimes forwarded from the remote server on which they are initially placed to a different location. Under the HTTP (Hypertext Transport Protocol), such forwarding is sometimes referred to as a "redirect." When an HTML document is initially stored on one remote server and then later transferred to another remote server, the first remote server will provide, in response to a request for that document, an indication that the document has been transferred to a new remote server. This indication generally includes a forwarding address ("redirect address"), which is generally a URL.

[0112] In the prior art, when a computer requesting a Web page receives a redirect, it must then submit a new request to the redirect address. Having to submit a second request and wait for a second response consumes time and increases overall latency. Consequently, the present invention uses the document database **61** to store any redirect address for each document or image. Any time a redirected document is requested, the server **5** automatically accesses the redirect address to retrieve the document. The document or image is provided to the client **1** based on only a single request from the client **1**. The change in location of the redirected document or image remains completely transparent to the client **1**.

[0113] **FIG. 9** illustrates a routine performed by the server **5** in accessing documents which may have been forwarded to a new remote server. Initially, the server **5** receives a request for a document, which generally includes an address (step **901**). The server **5** then accesses the document database **65** to determine whether there is a redirect address for the requested document (step **902**). If there is no redirect address, then the server **5** accesses a remote server **4** based on the address provided in the document request from the client **1** (step **903**). Assuming that the remote server **4** does not respond to the server **5** with a redirect (step **904**), the document is retrieved and downloaded to the client **1** by the server **5** (step **907**). If, however, a redirect address was stored in the document database **65** (step **902**), then the server **5** accesses the requested document according to the redirect address (step **906**). Or, if the remote server **4** responded with a redirect (step **904**), then the server **5** saves the redirect address to the document database **61** (step **905**) and accesses the requested document according to the redirect address (step **906**).

4. Other Proxy Functions

[0114] The document database **65** also stores information relating to the performance of each remote server **4** from which a document is retrieved. This information includes the latency and throughput of the remote server **4**. Such information can be valuable in instances where a remote server **4** has a history of responding slowly. For example, when the document is requested, this knowledge can be used by the server **5** to provide a predefined signal to the client **1**. The client **1** can, in response to the signal, indicate to the user that a delay is likely and give the user the option of canceling the request.

5. Backoff Mode

[0115] Although the server **5** generally operates in the proxy mode, it can also enter a "backoff mode" in which the server **5** does not act as a proxy, or the server **5** performs only certain aspects of the normal proxying functions. For example, if the proxy cache **65** is overloaded, then the server **5** can enter a backoff mode in which documents are not cached but are transcoded as required. Alternatively, during times when the server **5** is severely overloaded with network traffic, the server **5** may instruct the client **1** to bypass the server **5** and contact remote servers **4** directly for a specified

time or until further notice. Or, the server **5** can enter a flexible backoff mode in which the client 1 will be instructed to contact a remote server **4** directly only for certain Web sites for a limited period of time.

D. Access to WebTV™ Services

[0116] The WebTV™ server **5** provides various services to the client **1**, such as proxying and electronic mail ("e-mail"). In the prior art, certain difficulties are associated with allowing a client computer access to different services of an Internet service, as will now be explained with reference to **FIG. 10**.

[0117] **FIG. 10** illustrates a client-server system according to one prior art embodiment. The server **76** provides various services A, B, and C. The server **76** includes a database **71** for storing information on the user's access privileges to services A, B, and C. The client **75** of the embodiment of **FIG. 10** accesses any of services A, B, and C by contacting that service directly. The contacted service then accesses the database **71**, which stores the access privileges of the client **75**, to determine whether the client **75** should be allowed to access that service. Hence, each service provided by the server **76** requires direct access to the database **71**. This architecture results in a large number of accesses being made to the database **71**, which is undesirable. In addition, the fact that each service independently has access to the database **71** raises security concerns. Specifically, it can be difficult to isolate sensitive user information. The present invention overcomes such difficulties using a technique which is now described.

1. Tickets Containing Privileges And Capabilities

[0118] As shown in **FIG. 11**, the server **5** provides a number of services D, E, and F, and a log-in service **78**. The log-in service is used specifically to control initial log-on procedures by a client **1**. The log-in service **78** has exclusive access to the user database **62** (discussed above with respect to **FIG. 4B**). The log-in service **78** and the user database **62** are located within a first security zone **84**. Service D is located within a second security zone **86**, while services E and F are contained within a third security zone **88**. Note that the specific arrangement of security zones **84**, **86**, and **88** with respect to services D, E, and F is illustrative only.

[0119] The user database **66** of the present invention stores various information pertaining to each authorized user of a client **1**. This information includes account information, a list of the WebTV™ that services are available to the particular user, and certain user preferences. For example, a particular user may not wish his client **1** to be used to access Web pages having adult-oriented subject matter. Consequently, the user would request that his account be filtered to prevent access to such material. This request would then be stored as part of the user data in the user database **66**.

[0120] With regard to user preferences, the hypertext links selected by a given user can be tracked, and those having the largest number can be stored in the user database **66**. The list can then be provided to the client **1** for use in generating a menu screen of the user's favorite Web sites, to allow the user to directly access those Web sites. The list can also be used by the server **5** to analyze the user's interests and to formulate and provide to the user a list of new Web sites which the user is likely to be interested in. The list might be composed by associated key words in Web pages selected by the user with other Web pages.

[0121] Referring again to **FIG. 11**, in response to a log-on request by a client **1**, the log-in service **78** consults the user database **62** to determine if access to the server **5** by this particular client **1** is authorized. Assuming access is authorized, the log-in service **78** retrieves certain user information pertaining to this particular client **1** from the user database **62**. The log-in service then generates a "ticket"**82**, which is an information packet including the retrieved information. The ticket **82** is then provided to the client 1 which requested access.

[0122] The ticket **82** includes all information necessary to describe the access privileges of a particular user with respect to all services provided by the server 5. For example, the ticket may include the user name registered to the client **1**, the e-mail address assigned to client **1**, and any filtering requested by the user with respect to viewing Web sites. Each time the user requests access to one of the services D, E, or F, the client **1** submits a copy of the ticket **82** to that service. The requested service can then determine from the copy of the ticket **82** whether access to that service by that client **1** is authorized and, if so, any important information relating to such access.

[0123] None of the services provided by the server **5**, other than the log-in service **78**, has access to the user database **62**. Hence, any security-sensitive information can be isolated within the user database **62** and the log-in service **78**. Such isolation allows the individual services provided by the server **5** to be placed within separate "firewalls" (security regions), illustrated as security zones **84**, **86**, and **88**. In addition, this technique greatly reduces the number of accesses required to the user database **62** compared to the prior art embodiment illustrated in **FIG. 10**.

2. Redundancy of Services and Load Balancing

[0124] The present invention also includes certain redundancies in the various services provided by the server **5**. In particular, a given service (e.g., e-mail) can be provided by more than one physical or logical device. Each such device is considered a "provider" of that service. If a given provider is overloaded, or if the client **1** is unable to contact that provider, the client **1** can contact any of the other providers of that service. When the server **5** receives a log-in request from a client **1**, in addition to generating the above-described ticket **82**, the log-in service **78** dynamically generates a list of available WebTV™ services and provides this list to the client **1**.

[0125] The server **5** can update the list of services used by any client **1** to reflect services becoming unavailable or services coming on-line. Also, the list of services provided to each client **1** can be updated by the server **5** based upon changes in the loading of the server **5**, in order to optimize traffic on the server **5**. In addition, a client's list of services can be updated by services other than the log-in service **78**, such that one service can effectively introduce another service to the client **1**. For example, the e-mail service may provide a client **1** with the name, port number and IP of its address book service. Thus, one service can effectively, and securely within the same chain of trust, introduce another service to the client **1**.

[0126] This list of services includes the name of each service, a port number for the provider of each service, and an IP (Internet Protocol) for each service. Different providers of the same service are designated by the same name, but different port numbers and/or IPs. Note that in a standard URL, the protocol is normally specified at the beginning of the URL, such as "HTTP:/ /www . . . " under the HTTP protocol. However, according to the present invention, the normal protocol designation (i.e., "HTTP") in the URL is replaced with the name of the service, since the port number and IP for each service are known to the client **2**. Hence, the client **1** can access any of the redundant providers of a given service using the same URL. This procedure effectively adds a level of indirection to all accesses made to any WebTV™ service and automatically adds redundancy to the proxy service. It should also be noted that separate service names can also refer to the same service.

[0127] Assume, for example, that the e-mail service provided by the WebTV™ system is designated by the service name "WTV-mailto." A client **1** can access any provider of this e-mail service using the same URL. The client **1** merely chooses the appropriate port number and IP number to distinguish between providers. If the client **1** is unable to connect to one e-mail provider, it can simply contact the next one in the list.

[0128] Thus, at log-in time, a client **1** is provided with both a ticket containing privileges and capabilities as well as a list of service providers, as illustrated in **FIG. 12**. Initially, the log-in service **78** determines whether the user of client **1** is a valid user (step **1201**). If not, log-in is denied (step **1205**). If the user is a valid user, then the log-in service **78** gathers user information from the user database **62** and generates a ticket **82** (step **1202**). The log-in service **78** also generates the above-described list of services (step **1203**). The ticket **82** and the list of services are then downloaded to the client **1** (step **1204**).

3. Asynchronous Notification to Clients by Server

[0129] Another limitation associated with prior art Internet servers is the inability to provide asynchronous notification information to the client in the absence of a request from the client to do so. It would be desirable, for example, for a server to notify a client on its own initiative when a particular Web page has changed or that a particular service is inaccessible. The server **5** of the present invention provides such capability, and the client **1** is configured to receive and decode such notifications. For example, the client **1** can receive updates of its listing of service providers from the server **5** at various points in time, as already described. Similarly, if a particular service provider becomes unavailable, that fact will be automatically communicated to the client **1**. As another example, if e-mail addressed to the user has been received by the server **5**, then the server **5** will send a message to the client **1** indicating this fact. The client **1** will then notify the user that e-mail is waiting by a message displayed on the television set **12** or by an LED (light emitting diode) built into the housing of WebTV™ box **10**.

[0130] Thus, a method and apparatus have been described for providing proxying and transcoding of documents in a network. Although the present invention has been described with reference to specific exemplary embodiments, it will be evident that various modifications and changes may be made to these embodiments without departing from the broader spirit and scope of the invention as set forth in the claims. Accordingly, the specification and drawings are to be regarded in an illustrative rather than a restrictive sense.

What is claimed is:

1. In a proxying server coupled to a client, a method of providing a document to the client, the method comprising the steps of:

providing the document to the proxying server, the document including image data and non-image data for causing the client to generate a display;

partitioning the document into a plurality of partitions;

downloading data of a first partition to the client; and

repeating a step of downloading data of a next partition to the client, the step of downloading data of the next partition to the client being performed after a step of downloading data of a previous partition to the client, the repeating step repeated until each one of the plurality of partitions has been downloaded to the client.

2. The method described in claim 1 wherein the providing step includes the step of retrieving the document from a remote server coupled to the proxying server in response to a request from the client.

3. The method described in claim 1 wherein the step of downloading data of the first partition includes the steps of:

downloading non-image data of the first partition to the client; and

downloading image data of the first partition to the client.

4. The method described in claim 1 wherein the step of downloading data of the next partition includes the steps of:

downloading non-image data of the next partition to the client; and

downloading image data of the next partition to the client.

5. The method described in claim 1 wherein the partitioning step includes the steps of:

laying out a Web page caused to be generated by the document; and

partitioning the Web page into the plurality of partitions, each one of the plurality of partitions having a display height corresponding to a viewable display height of a screen of the client.

6. The method described in claim 5 wherein the document includes Hypertext Mark-up Language (HTML) data, wherein the step of downloading data of the first partition includes the steps of:

downloading HTML data of the first partition to the client; and

downloading image data of the first partition to the client.

7. The method described in claim 5 wherein the document includes Hypertext Mark-up Language (HTML) data, wherein the step of downloading data of the next partition includes the steps of:

downloading HTML data of the next partition to the client; and

downloading image data of the next partition to the client.

8. The method described in claim 5 wherein the client includes a television display, the viewable display height of

the screen of the client corresponding to a screen height of the television display in pixels.

9. In a proxying server coupled to a client and to a remote server, the proxying server operating as a proxy on behalf or the client for accessing the remote server, a method of providing a document to the client, the method comprising the steps of:

retrieving the document from the remote server in response to a request from the client, the document including Hypertext Mark-up Language (HTML) data and image data for causing the client to generate a Web page on a display;

downloading HTML data of a first partition of the document to the client, the first partition having a display height corresponding to a viewable display height of the display;

downloading image data of the first partition of the document to the client; and,

repeating, until the document has been entirely provided to the client, the steps of:

downloading HTML data of a next partition to the client, the next partition having a display height corresponding to the viewable display height of the display; and,

downloading image data of the next partition to the client.

10. The method described in claim 9 wherein the client includes a television display, the viewable display height of the display corresponding to a screen height of the television display in pixels.

11. The method described in claim 9 including the additional step of laying out the Web page caused to be generated by the document, the laying out step performed after the retrieving step.

12. A method of transmitting a document, the method comprising the steps of:

selecting a first remaining viewable portion of the document; and

reordering the document to transmit the first remaining viewable portion of the document prior to transmitting next remaining viewable portions of the document.

13. A method described in claim 12, wherein the step of selecting the viewable portion of the document includes the steps of:

laying out the document; and

partitioning the document into viewable portions.

14. A method described in claim 12 including the additional step of transmitting the next remaining viewable portions of the document sequentially until the document is transmitted.

* * * * *

US006470389B1

(12) **United States Patent**
Chung et al.

(10) **Patent No.: US 6,470,389 B1**
(45) **Date of Patent: *Oct. 22, 2002**

(54) **HOSTING A NETWORK SERVICE ON A CLUSTER OF SERVERS USING A SINGLE-ADDRESS IMAGE**

(75) Inventors: **Pi-Yu Chung**, Berkeley Heights, NJ (US); **Om P. Damani**, Austin, TX (US); **Yennun Huang**, Bridgewater, NJ (US); **Chandra M. Kintala**, Warren, NJ (US); **Yi-Min Wang**, Berkeley Heights, NJ (US)

(73) Assignees: **Lucent Technologies Inc.**, Murray Hill, NJ (US); **AT&T Corp.**, New York, NY (US)

(*) Notice: This patent issued on a continued prosecution application filed under 37 CFR 1.53(d), and is subject to the twenty year patent term provisions of 35 U.S.C. 154(a)(2).

Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **08/818,989**

(22) Filed: **Mar. 14, 1997**

(51) **Int. Cl.**[7] **G06F 15/16**

(52) **U.S. Cl.** **709/227**; 709/245

(58) **Field of Search** 395/674, 181, 395/200.33, 200.49, 200.65, 200.57, 200.58, 200.48; 370/401; 707/10, 227, 228, 218, 219, 235, 203, 5; 709/245

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,371,852 A 12/1994 Attanasio et al.
5,485,579 A * 1/1996 Hitz et al. 395/200.12
5,661,719 A * 8/1997 Townsend et al. 370/216

OTHER PUBLICATIONS

S.L. Garfinkel, "The Wizard of Netscape," Webserver Magazine, Jul./Aug. 1996, pp. 59–63.

"Hypertext Transfer Protocol—HTTP/1.0," Network Working Group, May 1996, <http://www.ics.uci.edu/pub/ietf/http>.

D. Anderson, T. Yang, V. Holmedahl and O.H. Ibarra, "SWEB: Towards a Scalable World Wide Web Server on Multicomputers," <http://www.cs.ucsb.edu/research/rapid_sweb/SWEB.html>.

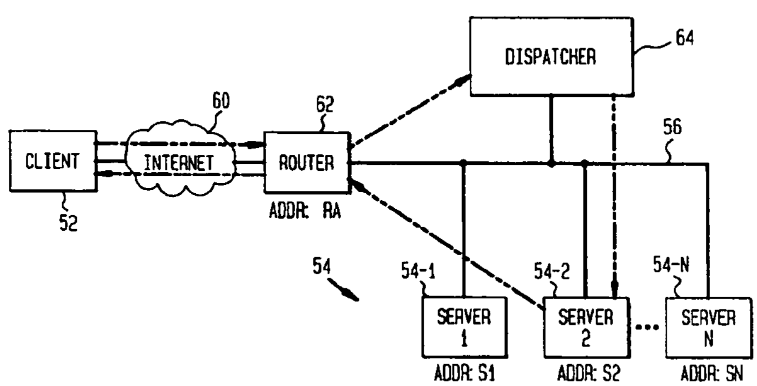
(List continued on next page.)

Primary Examiner—Moustafa M. Meky
Assistant Examiner—Tod Kupstas
(74) *Attorney, Agent, or Firm*—Ryan, Mason & Lewis, LLP

(57) **ABSTRACT**

Methods and apparatus for hosting a network service on a cluster of servers, each including a primary and a secondary Internet Protocol (IP) address. A common cluster address is assigned as the secondary address to each of the servers in the cluster. The cluster address may be assigned in UNIX-based servers using an ifconfig alias option, and may be a ghost IP address that is not used as a primary address by any server in the cluster. Client requests directed to the cluster address are dispatched such that only one of the servers of the cluster responds to a given client request. The dispatching may use a routing-based technique, in which all client requests directed to the cluster address are routed to a dispatcher connected to the local network of the server cluster. The dispatcher then applies a hash function to the client IP address in order to select one of the servers to process the request. The dispatching may alternatively use a broadcast-based technique, in which a router broadcasts client requests having the cluster address to all of the servers of the cluster over a local network. The servers then each provide a filtering routine, which may involve comparing a server identifier with a hash value generated from a client address, in order to ensure that only one server responds to each request broadcast by the router.

39 Claims, 3 Drawing Sheets



OTHER PUBLICATIONS

C. Yoshikawa, B. Chun, P. Eastham, A. Vahdat, T. Anderson and D. Culler, "Using Smart Clients to Build Scalable Services," Proceedings of USENIX 1997 Annual Technical Conference, Anaheim, California, Jan. 6–10, 1997.

T. Brisco, "DNS Support for Load Balancing," Network Working Group, RFC 1794, <http://andrew2.andrew.cmu.edu/rfc/rfc1794.html>.

T. Kwan, R. McGrath and D. Reed, NCSA's World Wide Web Server: Design and Perfomance, IEEE Computer, pp. 68–74, Nov. 1995.

D.M. Dias, W. Kish, R. Mukherjee and R. Tewari, "A Scalable and Highly Available Web Server," Proceedings of COMPCON '96, pp. 85–92, 1996.

E. Anderson, D. Patterson and E. Brewer, "The Magicrouter, an Application of Fast Packet Interposing," Symposium on Operating Systems Design and Implementation, OSDI, 1996, <http://www.cs.berkeley.edu/~eanders/magicrouter/osdi96–mr–submission/ps>.

Cisco Local Director, <http://www.cisco.com/warp/public/751/lodir/index.html>.

K. Egevang and P. Francis, "The IP Network Address Translator," Network Working Group, RFC 1631, <http://www.safety.net/rfc 1631.txt>.

"Two Servers, One Interface" <http://www.thesphere.com/~dlp/TwoServers/>.

W.R. Stevens, TCP/IP Illustrated, vol. 1, Ch. 4–6, pp. 53–83.

NetBSD Project, <http://www. NetBSD.org>.

W.R. Stevens, TCP/IP Illustrated, vol. 3, pp. 185–186.

Y. Huang and C. Kintala, "Software Implemented Fault Tolerance: Technologies and Experience," Proceedings of the 23rd International Symposium on Fault–Tolerant Computing—FTCS, Toulouse, France, pp. 2–9, Jun. 1993.

IBM Interactive Network Dispatcher (IND), http://www.ics.raleigh.ibm.com/netdispatch, "Interactive Network Dispatcher Version 1.1 for Sun Solaris and Microsoft Windows," "NT Join Interactive Session Support for AIX," and "Interactive Session Support for AIX Manages Interactive Load Distribution Across AIX Clusters," 6 pp.

* cited by examiner

FIG. 1
(PRIOR ART)

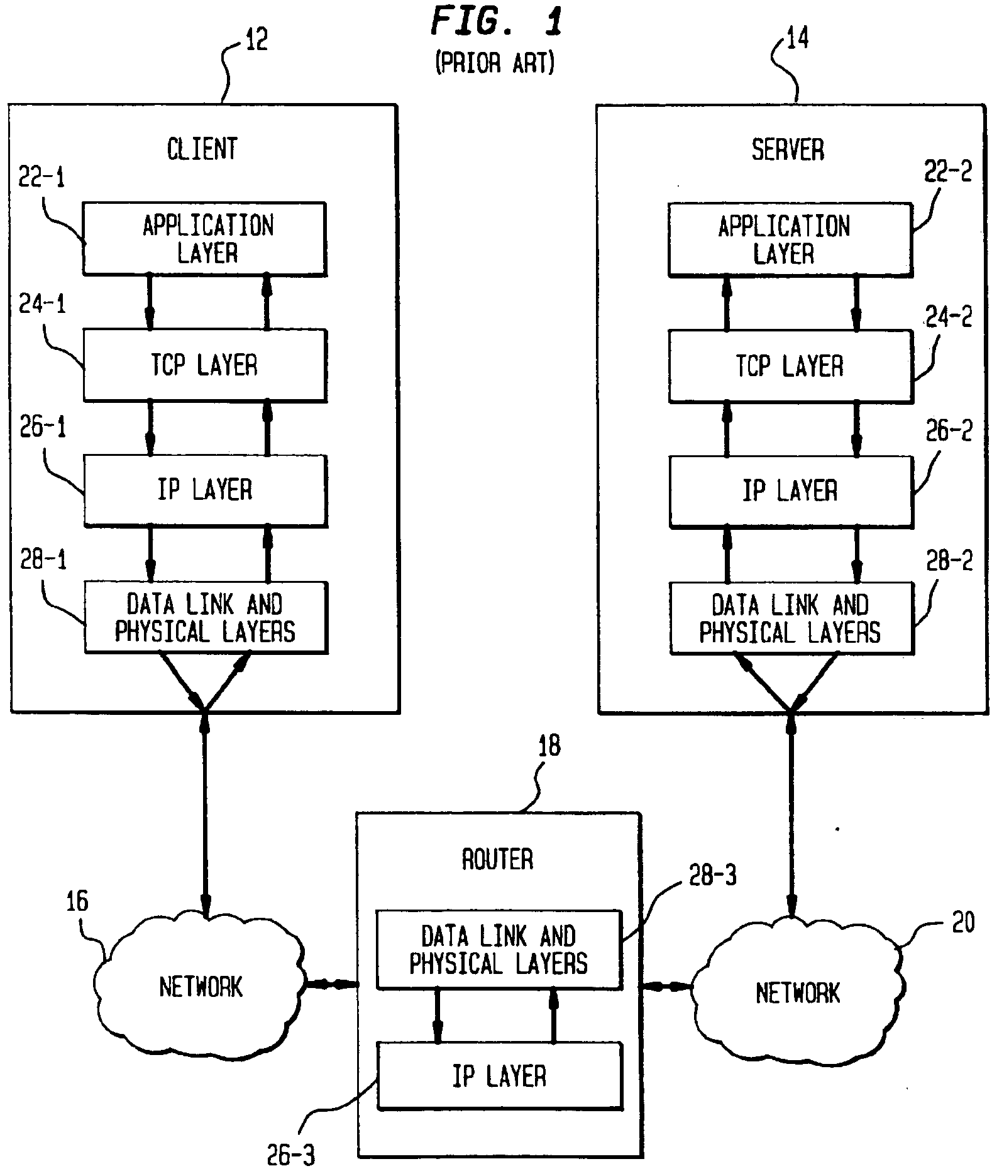


FIG. 2
(PRIOR ART)

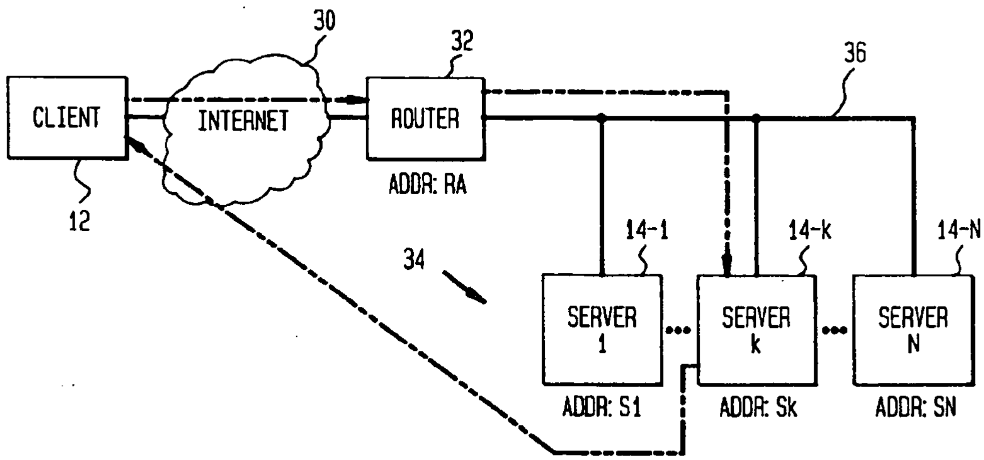


FIG. 3
(PRIOR ART)

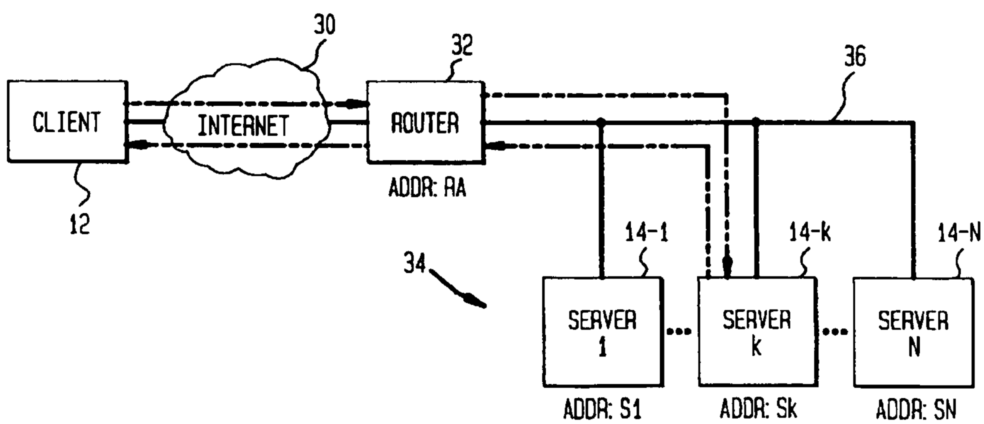


FIG. 4

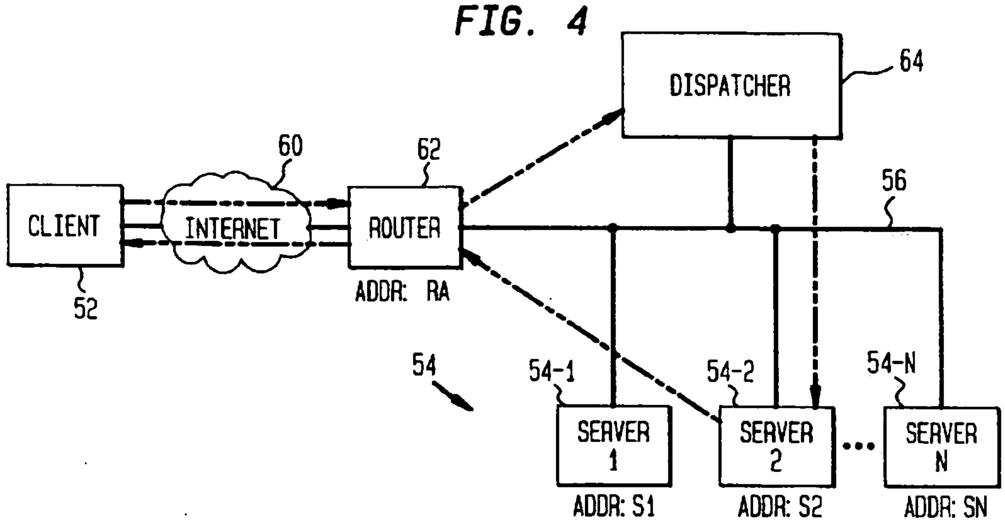
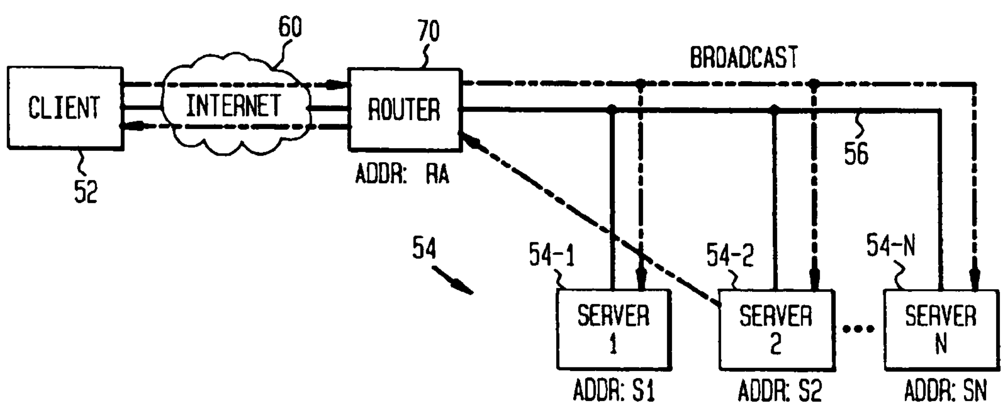

DISPATCHER
64
60
62
56
CLIENT
INTERNET
ROUTER
ADDR: RA
52
54
54-1
54-2
54-N
SERVER 1
SERVER 2
SERVER N
ADDR: S1
ADDR: S2
ADDR: SN

FIG. 5


60
70
BROADCAST
CLIENT
INTERNET
ROUTER
ADDR: RA
56
52
54
54-1
54-2
54-N
SERVER 1
SERVER 2
SERVER N
ADDR: S1
ADDR: S2
ADDR: SN

HOSTING A NETWORK SERVICE ON A CLUSTER OF SERVERS USING A SINGLE-ADDRESS IMAGE

FIELD OF THE INVENTION

The present invention relates generally to data communication networks such as the Internet and more particularly to techniques for hosting network services on a cluster of servers used to deliver data over a network in response to client requests, where the cluster of servers can be collectively identified by a client using a single-address image.

BACKGROUND OF THE INVENTION

With the explosive growth of the World Wide Web, many popular Internet web sites are heavily loaded with client requests. For example, it has been reported in S. L. Garfinkel, "The Wizard of Netscape," Webserver Magazine, July/August 1996, pp. 59–63, that home pages of Netscape Communications receive more than 80 million client requests or "hits" per day. A single server hosting a service is usually not sufficient to handle this type of aggressive growth. As a result, clients may experience slow response times and may be unable to access certain web sites. Upgrading the servers to more powerful machines may not always be cost-effective. Another common approach involves deploying a set of machines, also known as a cluster, and configuring the machines to work together to host a single service. Such a server cluster should preferably publicize only one server name for the entire cluster so that any configuration change inside the cluster does not affect client applications. The World Wide Web and other portions of the Internet utilize an application-level protocol, known as the Hypertext Transfer Protocol (HTTP), which is based on a client/server architecture. The HTTP protocol is described in greater detail in "Hypertext Transfer Protocol—HTTP/1.0," Network Working Group, May 1996, <http://www.ics.uci.edu/pub/ ietf/http>, which is incorporated by reference herein.

FIG. 1 illustrates an exemplary client/server architecture suitable for implementing HTTP-based network services on the Internet. A client **12** generates an HTTP request for a particular service, such as a request for information associated with a particular web site, and a Transmission Control Protocol/Internet Protocol (TCP/IP) connection is then established between the client **12** and a server **14** hosting the service. The client request is delivered to the server **14** in this example via a TCP/IP connection over a first network **16**, a router **18** and a second network **20**. The first network **16** may be a wide area communication network such as the Internet, while the second network **20** may be an Ethernet or other type of local area network (LAN) interconnecting server **14** with other servers in a server cluster. The router **18**, also referred to as a gateway, performs a relaying function between the first and second networks which is transparent to the client **12**.

The client request is generated by a web browser or other application-layer program operating in an application layer **22-1** of the client **12**, and is responded to by a file transfer system or other program in an application layer **22-2** of the server **14**. The requested network service may be designated by a Uniform Resource Locator (URL) which includes a domain name identifying the server **14** or a corresponding server cluster hosting the service. The application-level program of the client **12** initiates the TCP/IP connection by requesting a local or remote Domain Name Service (DNS) to map the server domain name to an IP address. The TCP and IP packet routing functions in client **12** and server **14** are provided in respective TCP layers **24-1**, **24-2** and IP layers **26-1**, **26-2**. The TCP and IP layers are generally associated with the transport and network layers, respectively, of the well-known Open Systems Interconnection (OSI) model. The TCP layers **24-1**, **24-2** process TCP packets of the client request and server response. The TCP packets each include a TCP header identifying a port number of the TCP connection between the client **12** and server **14**. The IP layers **26-1**, **26-2** process IP packets formed from the TCP packets of the TCP layers. The IP packets each include an IP header identifying an IP address of the TCP/IP connection between the client **12** and server **14**.

The IP address for a given network service may be determined, as noted above, by the client accessing a conventional DNS. The IP layer **26-1** of the client **12** uses the resulting IP address as a destination address in the IP packet headers of client request packets. The IP address together with the TCP port number provide the complete transport address for the HTTP server process. The client **12** and server **14** also include data link and physical layers **28-1** for performing framing and other operations to configure client request or reply packets for transmission over the networks **16** and **20**. The router **18** includes data link and physical layers **28-3** for converting client request and server reply packets to IP format, and an IP layer **26-3** for performing packet routing based on IP addresses. The server **14** responds to a given client request by supplying the requested information over the established TCP/IP connection in a number of reply packets. The TCP/IP connection is then closed.

There are many known techniques for distributing HTTP client requests to a cluster of servers. FIGS. 2 and 3 illustrate server-side single-IP-address image approaches which present a single IP address to the clients. An example of this approach is the TCP router approach described in D. M. Dias, W. Kish, R. Mukherjee and R. Tewari, "A Scalable and Highly Available Web Server," Proceedings of COMPCON '96, pp.85–92,1996, which is incorporated by reference herein. FIG. 2 illustrates the TCP router approach in which a client **12** establishes a TCP/IP connection over Internet **30** with a server-side router **32** having an IP address RA. The router **32** is connected via a LAN **36** to a server cluster **34** including N servers **14-i**, i =1, 2, . . . N, having respective IP addresses S1, S2, . . . SN. Each server of the cluster **34** generally provides access to the same set of contents, and the contents may be replicated on a local disk of each server, shared on a network file system, or served by a distributed file system.

The single-address image is achieved by publicizing the address RA of the server-side router **32** to the clients via the DNS. The client **12** therefore uses RA as a destination IP address in its request. The request is directed to the router **32**, which then dispatches the request to a selected server **14-k** of server cluster **34** based on load characteristics, as indicated by the dashed line connecting client **12** to server **14-k** via router **32**. The router **32** performs this dispatching function by changing the destination IP address of each incoming IP packet of a given client request from the router address RA to the address Sk of selected server **14-k**. The selected server **14-k** responds to the client request by sending reply packets over the established TCP/IP connection, as indicated by the dashed line connecting server **14-k** to client **12**. In order to make the TCP/IP connection appear seamless to the client **12**, the selected server **14-k** changes the source IP address in its reply packets from its address Sk to the

router address RA. The advantages of this approach are that it does not increase the number of TCP connections, and it is totally transparent to the clients. However, since the above-noted source IP address change is performed at the IP layer in a given server, the kernel code of every server in the cluster has to be modified to implement this mechanism. A proposed hybrid of the DNS approach and the TCP router approach, in which a DNS server selects one of several clusters of servers using a round-robin technique, suffers from the same problem.

FIG. 3 illustrates a server-side single-address image approach known as network address translation, as described in greater detail in E. Anderson, D. Patterson and E. Brewer, "The Magicrouter, an Application of Fast Packet Interposing," Symposium on Operating Systems Design and Implementation, OSDI, 1996, <http://www.cs.berkeley.edu/~eanders/magicrouter/osdi96-mr-submission.ps>, and Cisco Local Director, <http://www.cisco.com/warp/public/751/lodir/index. html>, which are incorporated by reference herein. As in the TCP router approach of FIG. 2, the client 12 uses the router address RA as a destination IP address in a client request, and the router 32 dispatches the request to a selected server 14-k by changing the destination IP address of each incoming request packet from the router address RA to the address Sk of selected server 14-k. However, in the network address translation approach, the source IP addresses in the reply packets from the selected server 14-k are changed not by server 14-k as in FIG. 2, but are instead changed by the router 32. The reply packet flow indicated by a dashed line in FIG. 2 thus passes from server 14-k to client 12 via router 32.

Compared to the TCP router approach of FIG. 2, network address translation has the advantage of server transparency. That is, no specific changes to the kernel code of the servers are required to implement the technique. However, both the TCP router and network address translation approaches require that the destination address in a request packet header be changed to a server address so that the server can accept the request. These approaches also require that the source address in a reply packet header be changed to the router address so that the client can accept the reply. These changes introduce additional processing overhead and unduly complicate the packet delivery process. In addition, because of the address changes, the above-described single-address image approaches may not be suitable for use with protocols that utilize IP addresses within an application, such as that described in K. Egevang and P. Francis, "The IP Network Address Translator," Network Working Group, RFC 1631, <http://www.safety.net/rfcl631.txt>, which is incorporated by reference herein. Furthermore, in both the TCP router and network address translation approaches, the router 32 needs to store an IP address mapping for every IP connection. Upon receiving an incoming packet associated with an existing TCP connection, the router has to search through all of the mappings to determine which server the packet should be forwarded to. The router itself may therefore become a bottleneck under heavy load conditions, necessitating the use of a more complex hardware design, as in the above-cited Cisco Local Director.

It is therefore apparent that a need exists for improved techniques for hosting a network service on a cluster of servers while presenting a single-address image to the clients, without the problems associated with the above-described conventional approaches.

SUMMARY OF THE INVENTION

The present invention provides methods and apparatus for hosting a network service on a cluster of servers. All of the servers in a server cluster configured in accordance with the invention may be designated by a single cluster address which is assigned as a secondary address to each server. All client requests for a web site or other network service associated with the cluster address are sent to the server cluster, and a dispatching mechanism is used to ensure that each client request is processed by only one server in the cluster. The dispatching may be configured to operate without increasing the number of TCP/IP connections required for each client request. The invention evenly distributes the client request load among the various servers of the cluster, masks the failure of any server or servers of the cluster by distributing client requests to the remaining servers without bringing down the service, and permits additional servers to be added to the cluster without bringing down the service. Although well-suited for use in hosting web site services, the techniques of the present invention may also be used to support a wide variety of other server applications.

In an exemplary embodiment of the invention, a network service is hosted by a server cluster in which each server includes a primary IP address and a secondary IP address. A common cluster address is assigned as the secondary IP address for each of the servers. The cluster address may be an IP address which does not correspond to a primary IP address of any of the servers. In UNIX-based servers, the cluster address may be assigned as the secondary address for a given server using an ifconfig alias option. If a given server includes multiple network interface cards, the cluster address may be assigned to one of the network interface cards using a UNIX ifconfig command without the alias option, or other similar technique. A router is coupled to a local network of the server cluster and is also coupled via the Internet to a client. The router receives client requests from the Internet, and uses a dispatching technique to direct client requests having the cluster address as a destination. The client requests are dispatched such that each of the requests is processed by only one of the servers in the cluster. The dispatching function may be based on the result of applying a hash function to an IP address of the given client. A suitable hash function may be determined using an analysis of a distribution of client IP addresses in an access log associated with one or more of the servers. In the event that a server has failed, the hash function may be reapplied to the client IP address to identify another server.

Two illustrative dispatching techniques for providing a single-address image for a server cluster in accordance with the invention include routing-based dispatching and broadcast-based dispatching. In the routing-based technique, a dispatcher is coupled to the router and to a local network of the server cluster. The router directs client requests having the cluster address to the dispatcher, and the dispatcher selects a particular one of the servers to process a given client request based on the result of applying a hash function to the client address. In the broadcast-based technique, the router broadcasts client requests having the cluster address to each of the servers over the local network of the server cluster. Each of the servers implements a filtering routine to ensure that each client request is processed by only one of the servers. The filtering routine may involve applying a hash function to the client IP address associated with a given client request, and comparing the result to a server identifier to determine whether that server should process the client request.

The techniques of the present invention provide fast dispatching and can be implemented with reduced cost and complexity. The techniques are suitable for use in TCP/IP networks as well as networks based on a variety of other

standards and protocols. Unlike the conventional single-address image approaches, the present invention does not require that a destination address in a request packet header be changed to a server address so that the server can accept the request, or that a source address in a reply packet header be changed to the router address so that the client can accept the reply. In addition, the router need not store an IP address mapping for every IP connection, nor is it required to search through such a mapping to determine which server a packet should be forwarded to. The router itself will therefore not become a bottleneck under heavy load conditions, and special router hardware designs are not required. These and other features and advantages of the present invention will become more apparent from the accompanying drawings and the following detailed description.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a block diagram illustrating a conventional client-server interconnection in accordance with the TCP/IP standard;

FIG. 2 illustrates a prior art TCP router technique for hosting a network service on a cluster of servers;

FIG. 3 illustrates a prior art network address translation technique for hosting a network service on a cluster of servers;

FIG. 4 illustrates a technique for hosting a network service on a cluster of servers using routing-based dispatching in accordance with an exemplary embodiment of the invention; and

FIG. 5 illustrates a technique for hosting a network service on a cluster of servers using broadcast-based dispatching in accordance with another exemplary embodiment of the invention.

DETAILED DESCRIPTION OF THE INVENTION

The present invention will be illustrated below in conjunction with exemplary client/ server connections established over the Internet to a server cluster using the Transmission Control Protocol/Internet Protocol (TCP/IP) standard. It should be understood, however, that the invention is not limited to use with any particular type of network or network communication protocol. The disclosed techniques are suitable for use with a wide variety of other networks and protocols. The term "server cluster" as used herein refers to a group or set of servers interconnected or otherwise configured to host a network service. The terms "cluster address" and "single-address image" refer generally to an address associated with a group of servers configured to support a network service or services. A "ghost IP address" is one type of cluster address in the form of an IP address which is not used as a primary address for any server of a given server cluster. The term "network service" is intended to include web sites, Internet sites and data delivery services, as well as any other data transfer mechanism accessible by a client over a network. The term "client request" refers to a communication from-a client which initiates the network service. A given client request may include multiple packets or only a single packet, depending on the nature of the request.

The present invention provides an improved single-address image approach to distributing client requests to servers of a server cluster. In a preferred embodiment, the invention allows all servers of a server cluster to share a single common IP address as a secondary address. The

secondary address is also referred to herein as a cluster address, and may be established using an ifconfig alias option available on most UNIX-based systems, or similar techniques available on other systems. The cluster address may be publicized to clients using the above-noted Domain Name Service (DNS) which translates domain names associated with Uniform Resource Locators (URLs) to IP addresses. All client requests to be directed to a service hosted by the server cluster are sent to the single cluster address, and dispatched to a selected one of the servers using routing-based or broadcast-based dispatching techniques to be described in greater detail below. Once a server is selected, future request packets associated with the same client request may be directed to the same server. All other communications within the server cluster may utilize primary IP addresses of the servers.

The above-noted ifconfig alias option is typically used to allow a single server to serve more than one domain name. For example, the ifconfig alias option allows a single server to attach multiple IP addresses, and thus multiple domain names, to a single network interface, as described in "Two Servers, One Interface" <http://www.thesphere.com/~dlp/TwoServers/>, which is incorporated by reference herein. Client requests directed to any of the multiple domain names can then be serviced by the same server. The server determines which domain name a given request is associated with by examining the destination address in the request packet. The present invention utilizes the ifconfig alias option to allow two servers to share the same IP address. Normally, two servers cannot share the same IP address because such an arrangement would cause any packet destined for the shared address to be accepted and responded to by both servers, confusing the client and possibly leading to a connection reset. Therefore, before a server is permitted to attach a new IP address to its network interface, a check may be made to ensure that no other server on the same local area network (LAN) is using that IP address. If a duplicate address is found, both servers are informed and warnings are issued. The routing-based or broadcast-based dispatching of the present invention ensures that every packet is processed by only one server of the cluster, such that the above-noted warnings do not create a problem.

An alternative technique for assigning a secondary address to a given server of a server cluster in accordance with the invention involves configuring the given server to include multiple network interface cards such that a different address can be assigned to each of the network interface cards. For example, in a UNIX-based system, conventional ifconfig commands may be used, without the above-described alias option, to assign a primary IP address to one of the network interface cards and a secondary IP address to another of the network interface cards. The secondary IP address is also assigned as a secondary IP address to the remaining servers in the cluster, and used as a cluster address for directing client requests to the cluster.

The exemplary embodiments of the present invention to be described below utilize dispatching techniques in which servers are selected based on a hash value of the client IP address. The hash value may be generated by applying a hash function to the client IP address, or by applying another suitable function to generate a hash value from the client IP address. For example, given N servers and a packet from a client having a client address CA, a dispatching function in accordance with the invention may compute a hash value k as CA mod (N−1) and select server k to process the packet. This ensures that all request or reply packets of the same TCP/IP connection are directed to the same server in the

server cluster. A suitable hash function may be determined by analyzing a distribution of client IP addresses in actual access logs associated with the servers such that client requests are approximately evenly distributed to all servers. When a server in the cluster fails, the subset of clients assigned to that server will not be able to connect to it. The present invention addresses this potential problem by dynamically modifying the dispatching function upon detection of a server failure. If the hash value of a given client IP address maps to the failed server, the client IP address is rehashed to map to a non-failed server, and the connections of the remaining clients are not affected by the failure.

FIG. 4 illustrates a routing-based dispatching technique in accordance with the present invention. Solid lines indicate network connections, while dashed lines show the path of an exemplary client request and the corresponding reply. A client 52 sends a client request to a server cluster 54 including N servers 54-i, i=1, 2, . . . N having IP addresses S1, S2, . . . SN and interconnected by an Ethernet or other type of LAN 56. The client request is formulated in accordance with the above-described HTTP protocol, and may include a URL with a domain name associated with a web site or other network service hosted by the server cluster 54. The client accesses a DNS to determine an IP address for the domain name of the service, and then uses the IP address to establish a TCP/IP connection for communicating with one of the servers 54-i of the server cluster 54. In accordance with the invention, a "ghost" IP address is publicized to the DNS as a cluster address for the server cluster 54. The ghost IP address is selected such that none of the servers 54-i of cluster 54 has that IP address as its primary address. Therefore, any request packets directed to the ghost IP address are associated with client requests for the service of the single-address image cluster 54. The use of the ghost IP address thus distinguishes a network service hosted by the cluster from activities of the servers 54-i which utilize the primary server addresses, and prevents interference with these primary address activities.

The client 52 uses the ghost IP address as a cluster address for directing its request to the server cluster 54. The request is directed over Internet 60 to a router 62 having an IP address RA. The router 62 includes a routing table having an entry or record directing any incoming request packets having the ghost IP address to a dispatcher 64 connected to the LAN 56. The dispatcher 64 includes an operating system configured to run in a router mode, using a routing algorithm which performs the dispatching described herein. In alternative embodiments, the functions of the dispatcher 64 could be incorporated into the router 62 in order to provide additional efficiency improvements. Each of the servers 54-i of the cluster 54 utilizes the above-described ifconfig alias option to set the ghost IP address as their secondary address. As noted above, this technique for setting a secondary address for each of the servers 54-i generally does not require any alteration of the kernel code running on the servers. In alternative embodiments, one or more of the servers 54-i may be configured to include multiple network interface cards, as previously noted, such that a different address can be assigned to each of the network interface cards of a given server using a UNIX ifconfig command or other similar technique.

The router 62 routes any packets having the ghost IP address to the dispatcher 64 in accordance with the above-noted routing table record. The dispatcher 64 then applies a hash function to the client IP address in a given request packet to determine which of the servers 54-i the given packet should be routed to. In the example illustrated in FIG. 4, the dispatcher 64 applies a hash function to the IP address of client 52 and determines that the corresponding request packet should be routed to server 54-2 at IP address S2. The dispatcher 64 then routes the request packet to the server 54-2 over LAN 56, as indicated by the dashed line, using the primary address S2 of server 54-2 to distinguish it from the other servers of cluster 54. After the network interface of server 54-2 accepts the packet, all higher level processing may be based on the ghost IP address because that is the destination address in the packet IP header and possibly in the application-layer packet contents. After processing the request, the server 54-2 replies directly to the client 52 via router 62 over the established TCP/IP connection, using the ghost IP address, and without passing through the dispatcher 64.

It should be noted that when a request packet destined for the ghost IP address is received by the network interface of dispatcher 64 and placed back onto the same network interface for delivery to one of the servers 54-i over LAN 56, it may cause an Internet control message protocol (ICMP) host redirect message to be sent to the router 62. This ICMP message is designed to direct the router 62 to update its routing table such that any future packets having the ghost IP address can bypass the dispatcher 64 and go directly to the destination server, as described in greater detail in W. R. Stevens, TCP/IP Illustrated, Vol. 1, Ch. 6, pp. 69–83, which is incorporated by reference herein. However, this effect is undesirable in the routing technique of FIG. 4 because the dispatcher 64 performs the server selection process as previously described. It therefore may be necessary to suppress the ICMP host redirect message for the ghost IP address by, for example, removing or altering the corresponding operating system code in the dispatcher. In the above-mentioned alternative embodiments in which the dispatching function is implemented within the router 62, the ICMP redirect message is not generated and therefore need not be suppressed. Another potential problem may arise when a reply packet is sent back to the client 52 from the selected server 54-2 with the ghost IP address, in that it may cause the router 62 to associate, in its Address Resolution Protocol (ARP) cache, the ghost IP address with the LAN address of the selected server. The operation of the ARP cache is described in greater detail in W. R. Stevens, TCP/IP Illustrated, Vol. 1, Chs. 4 and 5, pp. 53–68, which is incorporated by reference herein. The illustrative embodiment of FIG. 4 avoids this problem by automatically routing the request packets to the dispatcher 64, and then dispatching based on the server primary IP address, such that the router ARP cache is not used.

FIG. 5 illustrates a broadcast-based dispatching technique in accordance with the present invention. Again, solid lines indicate network connections, while dashed lines show the path of an exemplary client request and the corresponding reply. As in the FIG. 4 routing-based embodiment, client 52 sends a client request to server cluster 54 including N servers 54-i, i =1, 2, . . . N connected to LAN 56 and having IP addresses S1, S2, . . . SN. The client 52 uses the above-described ghost address as a cluster address for directing its request to the server cluster 54. The request is directed over Internet 60 to a router 70 having an IP address RA. The router 70 broadcasts any incoming request packets having the ghost IP address to the LAN 56 interconnecting the servers 54-i of the server cluster 54, such that the request packet is received by each of the servers 54-i.

Each of the servers 54-i of the cluster 54 implements a filtering routine in order to ensure that only one of the servers 54-i processes a given client request. The filtering

routine may be added to a device driver of each of the servers **54**-i. In an exemplary implementation, each of the servers **54**-i is assigned a unique identification (ID) number. The filtering routine of a given server **54**-i computes a hash value of the client IP address and compares it to the ID number of the given server. If the hash value and the ID number do not match, the filtering routine of the given server rejects the packet. If the hash value and the ID number do match, the given server accepts and processes the packet as if it had received the packet through a conventional IP routing mechanism. In the illustrative example of FIG. **5**, a packet associated with the request from client **52** is broadcast by the router **70** to each of the servers **54**-i of the server cluster **54** over the LAN **56** as previously noted. The filtering routine of server **54-2** generates a hash value of the client IP address which matches the unique ID number associated with server **14-2**, and server **14-2** therefore accepts and processes the packet. The filtering routines of the N−1 other servers **54**-i each indicate no match between the client IP address and the corresponding server ID number, and therefore discard the broadcast packet. The reply packets are sent back to the client **52** via router **70**, as indicated by the dashed lines, using the ghost IP address.

The broadcast-based dispatching technique of FIG. **5** may be implemented using a permanent ARP entry within the router **70**, to associate the ghost IP address with the Ethernet or other local network broadcast address associated with LAN **56** of the cluster **54**. A potential problem is that any reply packet from a selected server appears to be coming from the ghost IP address, and may therefore cause the router **70** to overwrite the entry in its ARP cache such that the ghost IP address is associated with the LAN address of the selected server. This potential problem may be addressed by setting up a routing table entry in the router **70** to direct all packets having a ghost IP destination address to a second ghost IP address which is a legal subnet address in the LAN **56** of the server cluster **54** but is not used by any server. In addition, an entry is inserted in the ARP cache of the router **70** to associate the second ghost IP address with the broadcast address of the LAN **56**. When the router **70** routes a packet to the second ghost IP address, it will then actually broadcast the packet to each of the servers **54**-i of the cluster **54**. Since no reply packet is sent from the second ghost IP address, the corresponding entry of the router ARP cache will remain unchanged. Another potential problem is that some operating systems, such as the NetBSD operating system, do not allow a TCP packet to be processed if it is received from a broadcast address. This potential problem may be avoided by a suitable modification to the broadcast address in the LAN packet header attached to the packet.

The routing-based and broadcast-based dispatching techniques described in conjunction with FIGS. **4** and **5** above have been implemented on a cluster of Sun SPARC workstations. The NetBSD operating system, as described in NetBSD Project, <http://www. NetBSD.org>, was used to provide any needed kernel code modifications. The dispatching overhead associated with both techniques is minimal because the packet dispatching is based on simple IP address hashing, without the need for storing or searching any address-mapping information. In the routing-based dispatching technique, the additional routing step in the dispatcher **64** typically adds a delay of about 1 to 2 msecs to the TCP round-trip time of each incoming request packet. A study in W. R. Stevens, TCP/IP Illustrated, Volume 3, pp. 185–186, which is incorporated by reference herein, indicates that the median TCP round-trip time is 187 msecs. The additional delay attributable to the routing-based dispatching is therefore negligible. Although the additional routing step for every request packet sent to the ghost IP address may increase the traffic in the LAN of the server cluster, the size of a request in many important applications is typically much smaller than that of the corresponding response, which is delivered directly to the client without the additional routing. In the broadcast-based dispatching technique, the broadcasting of each incoming request packet on the LAN of the server cluster does not substantially increase network traffic. Although a hash value is computed for each incoming packet having the ghost IP destination address, which increases the CPU load of each server, this additional computation overhead is negligible relative to the corresponding communication delay.

Both the routing-based and broadcast-based dispatching techniques of the present invention are scalable to support relatively large numbers of servers. Although the dispatcher in the routing-based technique could present a potential bottleneck in certain applications, a study in the above-cited D. M. Dias et al. reference indicates that a single dispatcher can support up to 75 server nodes, which is sufficient support for many practical systems. The number of servers supported may be even higher with the present invention given that the routing-based dispatching functions described herein are generally simpler than those in the D. M. Dias et al. reference. It should also be noted that additional scalability can be obtained by combining the routing-based dispatching of the present invention with a DNS round-robin technique. For example, a DNS server may be used to map a domain name to one of a number of different ghost IP addresses belonging to different server clusters using a round-robin technique. In the broadcast-based dispatching technique, there is no potential dispatching bottleneck, although the device drivers or other portions of the servers may need to be modified to provide the above-described filtering routines.

The routing-based and broadcast-based dispatching of the present invention can also provide load balancing and failure handling capabilities. For example, given N servers and a packet from client address CA, the above-described routing-based dispatching function may compute a hash value k as CA mod (N−1) and select server k to process the packet. More sophisticated dispatching functions can also be used, and may involve analyzing the actual service access log to provide more effective load balancing. In order to detect failures, each server may be monitored by a watchdog daemon such as the watchd daemon described in greater detail in Y. Huang and C. Kintala, "Software Implemented Fault Tolerance: Technologies and Experience," Proceedings of the 23rd International Symposium on Fault-Tolerant Computing—FTCS, Toulouse, France, pp. 2–9, June 1993, which is incorporated by reference herein. When a server fails, the corresponding watchd daemon initiates a change of the dispatching function to mask the failure and rebalance the load. A system call interface may be implemented to allow the dispatching function to be changed while the servers remain on-line. In routing-based dispatching, the watchd daemon may notify the dispatcher to change the dispatching function, while in broadcast-based dispatching, all servers may be notified to modify their filtering routines. For example, if a server k fails, the new dispatching function may check to see if the hash value CA mod N equals k. If it does, a new hash value j=CA mod (N−1) is computed. If j is less than k, the packet goes to server j. Otherwise, the packet goes to server j+1. This technique does not affect the clients of non-failed servers, reassigns the clients of the failed server evenly to the remaining servers, and can be

readily extended to handle multiple server failures. Additional servers can be added to the cluster without bringing down the service by changing the dispatching function from CA mod N to CA mod (N+1).

In routing-based dispatching, the dispatcher may become a single point of failure, and therefore should also be monitored by a watchd daemon or other suitable failure monitoring mechanism. Upon detecting a failure, the watchd daemon may trigger a transfer of the dispatching function from the primary dispatcher to a backup dispatcher, and then direct the router to change the entry in its routing table such that future incoming request packets are routed to the backup dispatcher. Since no mapping table is maintained by the primary dispatcher, this approach is substantially stateless. Proper routing may be ensured by simply utilizing consistent routing functions in the primary and backup dispatchers, without the substantial additional costs associated with mapping-based approaches.

The use of the ifconfig alias option or other similar technique to provide a single-address image for a server cluster provides a number of advantages over the conventional techniques described previously. For example, it avoids the need to change the destination address in a request packet header so that a particular server can accept the request, and the need to change the source address in a reply packet header to the cluster address so that the client can accept the reply. With the single-address image approach of the present invention, all servers can accept and respond to packets having the cluster address, so that the addresses in the request and reply packet headers do not need to be modified. Since the single-image approach of the present invention does not require alteration of the packet addresses, it is suitable for use with a wide variety of protocols, including those protocols which utilize IP addresses within an application program. In addition, the single-address image approach of the present invention does not require a router to store or to search through a potentially large number of IP address mappings in order to determine which cluster server should receive a request packet. The invention thus effectively removes the possibility that the router may become a bottleneck under heavy load conditions.

The above-described embodiments of the invention are intended to be illustrative only. Numerous alternative embodiments may be devised by those skilled in the art without departing from the scope of the following claims.

What is claimed is:

1. A method of routing client requests to a plurality of servers configured to support a network service over a communication network, each of the servers having a primary address, the method comprising the steps of:

assigning a common address as a secondary address for each of the plurality of servers such that each of the servers individually has the secondary address; and

processing client requests directed to the common address such that each of the requests is processed by a particular one of the plurality of servers without modification of the common address in the request.

2. The method of claim 1 wherein the network utilizes a TCP/IP protocol and the primary and secondary addresses are primary and secondary IP addresses, respectively.

3. The method of claim 2 wherein the common address is an IP address which does not correspond to a primary IP address of any of the plurality of servers.

4. The method of claim 1 wherein at least one of the plurality of servers is a UNIX-based server including multiple network interface cards, and the assigning step includes assigning the common address for the at least one server using an ifconfig command.

5. The method of claim 1 wherein the plurality of servers are UNIX-based servers, and the assigning step includes assigning the common address utilizing an ifconfig alias option for at least a subset of the plurality of servers.

6. The method of claim 1 wherein the processing step includes the step of dispatching a request of a given client to one of the plurality of servers based on application of a hash function to an IP address of the given client.

7. The method of claim 6 wherein the hash function is determined based on an analysis of a distribution of client IP addresses in an access log associated with one or more of the servers.

8. The method of claim 6 wherein the dispatching step includes reapplying the hash function to the client IP address to identify another server if a server identified as a result of a previous application of the hash function has failed.

9. The method of claim 1 wherein the processing step includes the steps of:

routing client requests directed to the common address to a dispatcher connected to a local network associated with the plurality of servers; and

selecting a particular one of the servers to process a given client request based on application of a hash function to a corresponding client address in the dispatcher.

10. The method of claim 1 wherein the processing step includes the steps of:

broadcasting a given client request directed to the common address to each of the plurality of servers over a local network associated with the servers; and

implementing a filtering routine in each of the plurality of servers so that the given client request is processed by only one of the servers.

11. The method of claim 10 wherein the implementing step includes the steps of:

applying a hash function to a client IP address associated with the given client request; and

comparing the result of the applying step to an identifier of a particular server to determine whether that server should process the given client request.

12. An apparatus for routing client requests to a plurality of servers configured to support a network service over a communication network, each of the servers having a primary address, the apparatus comprising:

means for assigning a common address as a secondary address for each of the plurality of servers such that each of the servers individually has the secondary address; and

means for processing client requests directed to the common address such that each of the requests is processed by a particular one of the plurality of servers without modification of the common address in the request.

13. The apparatus of claim 12 wherein the network utilizes a TCP/IP protocol and the primary and secondary addresses are primary and secondary IP addresses, respectively.

14. The apparatus of claim 13 wherein the common address is an IP address which does not correspond to a primary IP address of any of the plurality of servers.

15. The apparatus of claim 12 wherein at least one of the plurality of servers is a UNIX-based server including multiple network interface cards, and the assigning means is operative to assign the common address for the at least one server using an ifconfig command.

16. The apparatus of claim 12 wherein the plurality of servers are UNIX-based servers, and the assigning means is

operative to assign the common address utilizing an ifconfig alias option for at least a subset of the plurality of servers.

17. The apparatus of claim 12 wherein the processing means is operative to dispatch a request of a given client to one of the plurality of servers based on application of a hash function to an IP address of the given client.

18. The apparatus of claim 17 wherein the hash function is determined based on an analysis of a distribution of client IP addresses in an access log associated with one or more of the servers.

19. The apparatus of claim 17 wherein the processing means is further operative to reapply the hash function to the client IP address to identify another server if a server identified as a result of a previous application of the hash function has failed.

20. The apparatus of claim 12 wherein the processing means further includes a dispatcher connected to a local network associated with the plurality of servers, wherein the dispatcher is operative to receive client requests directed to the common address, and to select a particular one of the servers to process a given client request based on application of a hash function to a corresponding client address.

21. The apparatus of claim 12 wherein the processing means further includes:

means for broadcasting a given client request directed to the common address to each of the plurality of servers over a local network associated with the servers; and

means for filtering the given client request in each of the plurality of servers so that the given client request is processed by only one of the servers.

22. The apparatus of claim 21 wherein the filtering means is operative to apply a hash function to a client IP address associated with the given client request, and to compare the result of the applying step to an identifier of a particular server to determine whether that server should process the given client request.

23. An apparatus for routing client requests for a network service over a communication network, the apparatus comprising:

a plurality of servers configured to support the network service, each of the servers having a primary address and a secondary address, wherein a common address is assigned as the secondary address for each of the plurality of servers such that each of the servers individually has the secondary address; and

a router coupled to the servers and operative to route client requests directed to the common address such that each of the requests is processed by a particular one of the plurality of servers without modification of the common address in the request.

24. The apparatus of claim 23 wherein the network utilizes a TCP/IP protocol and the primary and secondary addresses are primary and secondary IP addresses, respectively.

25. The apparatus of claim 24 wherein the common address is an IP address which does not correspond to a primary IP address of any of the plurality of servers.

26. The apparatus of claim 23 wherein at least one of the plurality of servers is a UNIX-based server including multiple network interface cards, and the common address is assigned for the at least one server using an ifconfig command.

27. The apparatus of claim 23 wherein the plurality of servers are UNIX-based servers, and the common address is assigned as the secondary address of the plurality of servers by utilizing an ifconfig alias option for at least a subset of the plurality of servers.

28. The apparatus of claim 23 wherein the router is further operative to route client requests such that a request of a given client is routed to one of the plurality of servers based on application of a hash function to an IP address of the given client.

29. The apparatus of claim 28 wherein the hash function is determined based on an analysis of a distribution of client IP addresses in an access log associated with one or more of the servers.

30. The apparatus of claim 28 wherein the hash function is reapplied to the client IP address to identify another server if a server identified as a result of a previous application of the hash function has failed.

31. The apparatus of claim 23 further including a dispatcher coupled to the router and to a local network associated with the plurality of servers, such that the router directs client requests having the common address to the dispatcher, and the dispatcher selects a particular one of the servers to process a given client request based on application of a hash function to a corresponding client address.

32. The apparatus of claim 23 wherein the router is further operative to broadcast a given client request directed to the common address to each of the plurality of servers over a local network associated with the servers, and further wherein each of the servers implements a filtering routine so that the given client request is processed by only one of the servers.

33. The apparatus of claim 32 wherein the filtering routine involves applying a hash function to a client IP address associated with the given client request, and comparing the result to an identifier of a particular server to determine whether that server should process the given client request.

34. A method of routing client requests to a plurality of servers configured to support a network service over a communication network, each of the servers having a primary address, the method comprising the steps of:

assigning a common address as a secondary address for each of the plurality of servers such that each of the servers individually has the secondary address; and

processing client requests directed to the common address such that each of the requests is processed by a particular one of the plurality of servers without modification of the common address in the request, wherein each of the servers provides access to substantially the same set of contents associated with the network service.

35. An apparatus for routing client requests to a plurality of servers configured to support a network service over a communication network, each of the servers having a primary address, the apparatus comprising:

means for assigning a common address as a secondary address for each of the plurality of servers such that each of the servers individually has the secondary address; and

means for processing client requests directed to the common address such that each of the requests is processed by a particular one of the plurality of servers without modification of the common address in the request, wherein each of the servers provides access to substantially the same set of contents associated with the network service.

36. An apparatus for routing client requests for a network service over a communication network, the apparatus comprising:

a plurality of servers configured to support the network service, each of the servers having a primary address

and a secondary address, wherein a common address is assigned as the secondary address for each of the plurality of servers such that each of the servers individually has the secondary address, and wherein each of the servers provides access to substantially the same set of contents associated with the network service; and

a router coupled to the servers and operative to route client requests directed to the common address such that each of the requests is processed by a particular one of the plurality of servers without modification of the common address in the request.

37. A method of routing client requests to a plurality of servers configured to support a network service over a communication network, each of the servers having a primary address, the method comprising the steps of:

assigning a common address as a secondary address for each of the plurality of servers such that each of the servers individually has the secondary address;

broadcasting a given client request directed to the common address to each of the plurality of servers; and

implementing a filtering routine in each of the plurality of servers so that the given client request is processed by only one of the servers without modification of the common address in the request.

38. An apparatus for routing client requests to a plurality of servers configured to support a network service over a communication network, each of the servers having a primary address, the apparatus comprising:

means for assigning a common address as a secondary address for each of the plurality of servers such that each of the servers individually has the secondary address;

means for broadcasting a given client request directed to the common address to each of the plurality of servers; and

means for filtering the given client request in each of the plurality of servers so that the given client request is processed by only one of the servers without modification of the common address in the request.

39. An apparatus for routing client requests for a network service over a communication network, the apparatus comprising:

a plurality of servers configured to support the network service, each of the servers having a primary address and a secondary address, wherein a common address is assigned as the secondary address for each of the plurality of servers such that each of the servers individually has the secondary address; and

a router coupled to the servers and operative to route client requests directed to the common address such that each of the requests is processed by a particular one of the plurality of servers, wherein the router is further operative to broadcast a given client request directed to the common address to each of the plurality of servers, and further wherein each of the servers implements a filtering routine so that the given client request is processed by only one of the servers without modification of the common address in the request.

* * * * *

US006314465B1

(12) **United States Patent**
Paul et al.

(10) **Patent No.: US 6,314,465 B1**
(45) **Date of Patent: Nov. 6, 2001**

(54) **METHOD AND APPARATUS FOR LOAD SHARING ON A WIDE AREA NETWORK**

(75) Inventors: **Sanjoy Paul**, Marlboro; **Sampath Rangarajan**, Bridgewater, both of NJ (US)

(73) Assignee: **Lucent Technologies Inc.**, Murray Hill, NJ (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/266,339**

(22) Filed: **Mar. 11, 1999**

(51) **Int. Cl.**[7] **G06F 13/00**

(52) **U.S. Cl.** **709/226**; 709/217

(58) **Field of Search** 709/201, 202, 709/203, 218, 226, 227, 223, 229, 217, 219

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,031,089	*	7/1991	Liu et al.	709/226
5,329,619	*	7/1994	Page et al.	709/203
5,495,426	*	2/1996	Waclawsky et al.	709/226
5,572,643	*	11/1996	Judson	709/218
5,644,720	*	7/1997	Boll et al.	709/227
5,774,660	*	6/1998	Brendel et al.	709/201
5,828,837	*	10/1998	Eikeland	709/202
5,933,606	*	8/1999	Mayhew	709/201
6,006,259	*	12/1999	Adelman et al.	709/223
6,070,191	*	5/2000	Narendran et al.	709/226
6,070,192	*	5/2000	Holt et al.	709/227
6,192,408	*	2/2001	Vahalia et al.	709/229

OTHER PUBLICATIONS

G. Goldszmidt et al., NetDispatcher: A TCP Connection Router, *IBM Research Report*, IBM Research Division, T.J. Watson Research Center, Yorktown Heights, NY, May 19, 1997, vol. RC 20853, p. 1–31.*

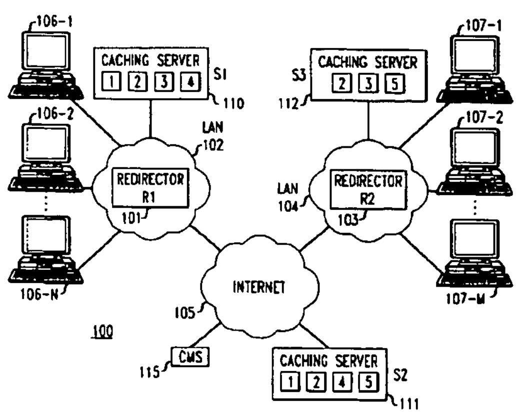
(List continued on next page.)

Primary Examiner—Glenton B. Burgess
Assistant Examiner—Seyed M. Safavian
(74) *Attorney, Agent, or Firm*—Stephen M. Gurey

(57) **ABSTRACT**

Client's (**106-1–106-N**, **107-1–107-M**) on local area networks (**102, 103**) making requests to hot sites, which are connected on a wide area network (**100**) such as the Internet, are redirected through one of a possible plurality of different redirectors (**101, 103**) to one of a possible plurality of caching servers (S1, S2, S3), which each have responsibility for mapping one or more of the hot sites. Each request is probabilistically directed by one of the redirectors to one of the caching servers that map the requested hot site in accordance with weights that are determined for that redirector-hot site pair so as to minimize the average delay that all client requests across the network will encounter in making requests to all the cached hot sites. In order to determine the weights with which each redirector will redirect requests to the hot sites to the caching servers, statistics of access rates to each hot site are dynamically determined by each redirector in the network from the traffic flow and reported to a central management station (CMS) (**115**). Network delay is similarly measured by each redirector and reported to the CMS, and server delay is computed using a queuing model of each server. Using these parameters as inputs, a non-linear programming optimization problem is solved as a network flow problem in order to determine the weights for each redirector that will minimize the average delay. As the access rate statistics, as well as the network delay and server delay, dynamically change, the CMS, using the network flow algorithm, recalculates the weights and forwards them back to each redirector. In other embodiments, the redirector-logical item pair for which the redirector probabilistically directs client requests may be other than a hot site identity. For example, the logical items can be groups of clients or groups of documents, and the servers to which requests are forwarded can be web servers or caching servers.

83 Claims, 5 Drawing Sheets



02/24/2004, EAST Version: 1.4.1

OTHER PUBLICATIONS

A. Bestavros., "Speculative Data Dissemination and Service to Reduce Server Load, Network Traffic and Service Time in Distributed Information System". 1063-6382 IEEE 1996. pp. 180-187.*

Antoine Mourad., "Scalable Web Server Architectures". 0-8186-7852-6/97 IEEE 1997. pp. 12-16.*

Azar Bestavros., "WWW Traffic Reduction and Load Balancing through Server-Based Caching". 1063-6552/97 IEEE 1997 pp. 56-67.*

Shiv Kalyanaraman et al., Performance of TCP/IP over ABR Service on ATM Networks. 0-7803-3336-5/96 IEEE 1996. pp. 468-475.*

* cited by examiner

FIG. 1

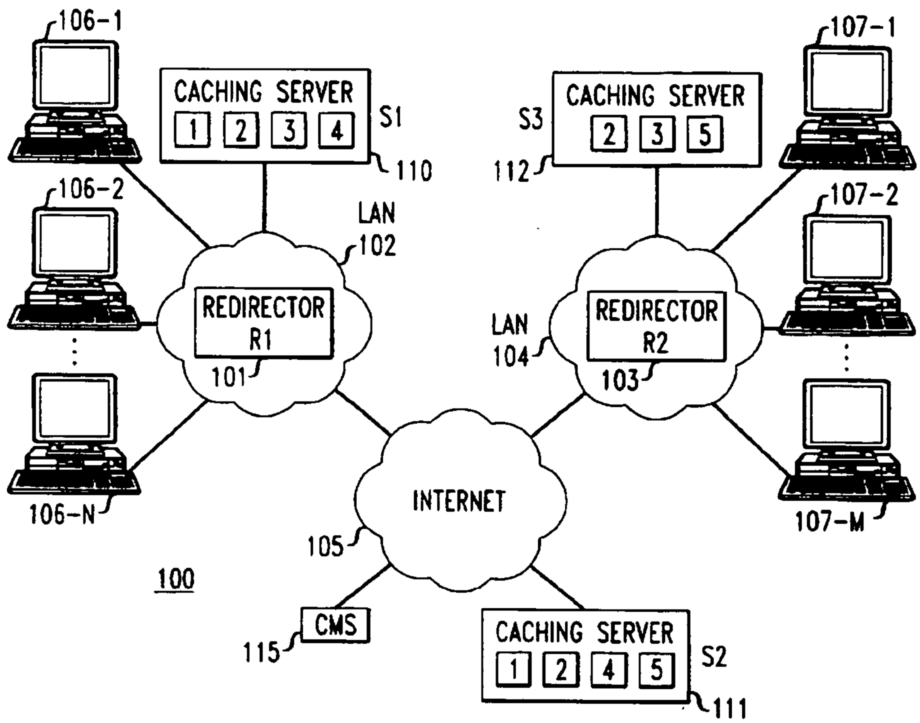


FIG. 2
(PRIOR ART)

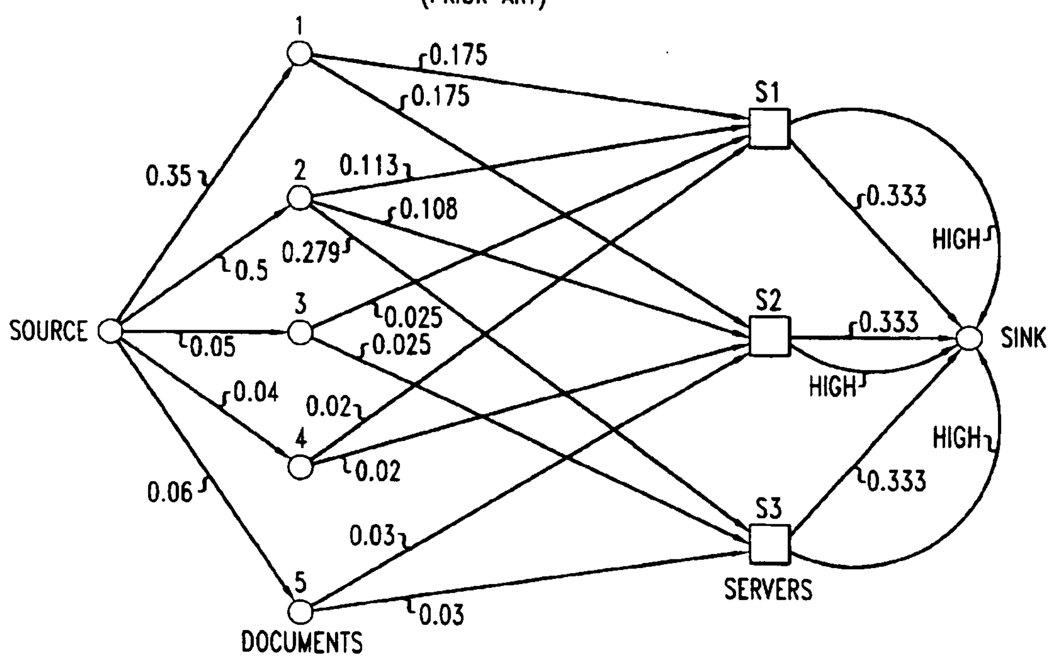
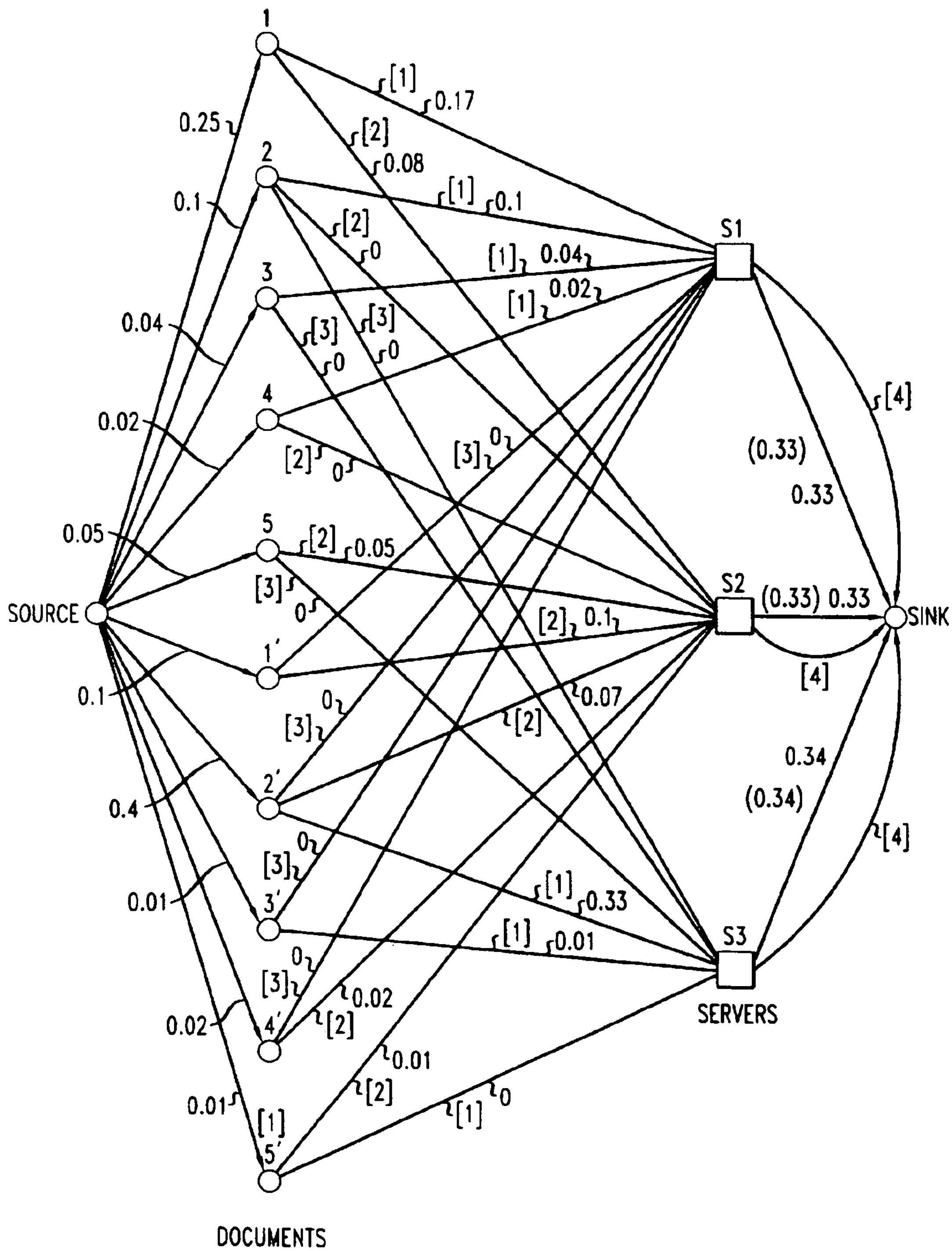
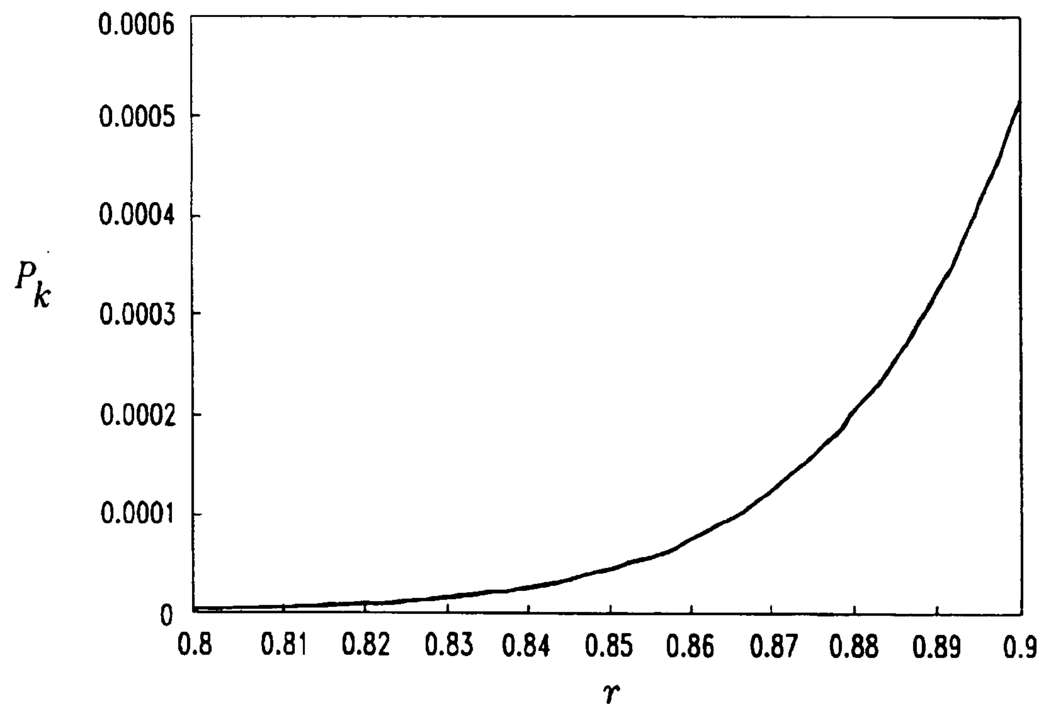


FIG. 3



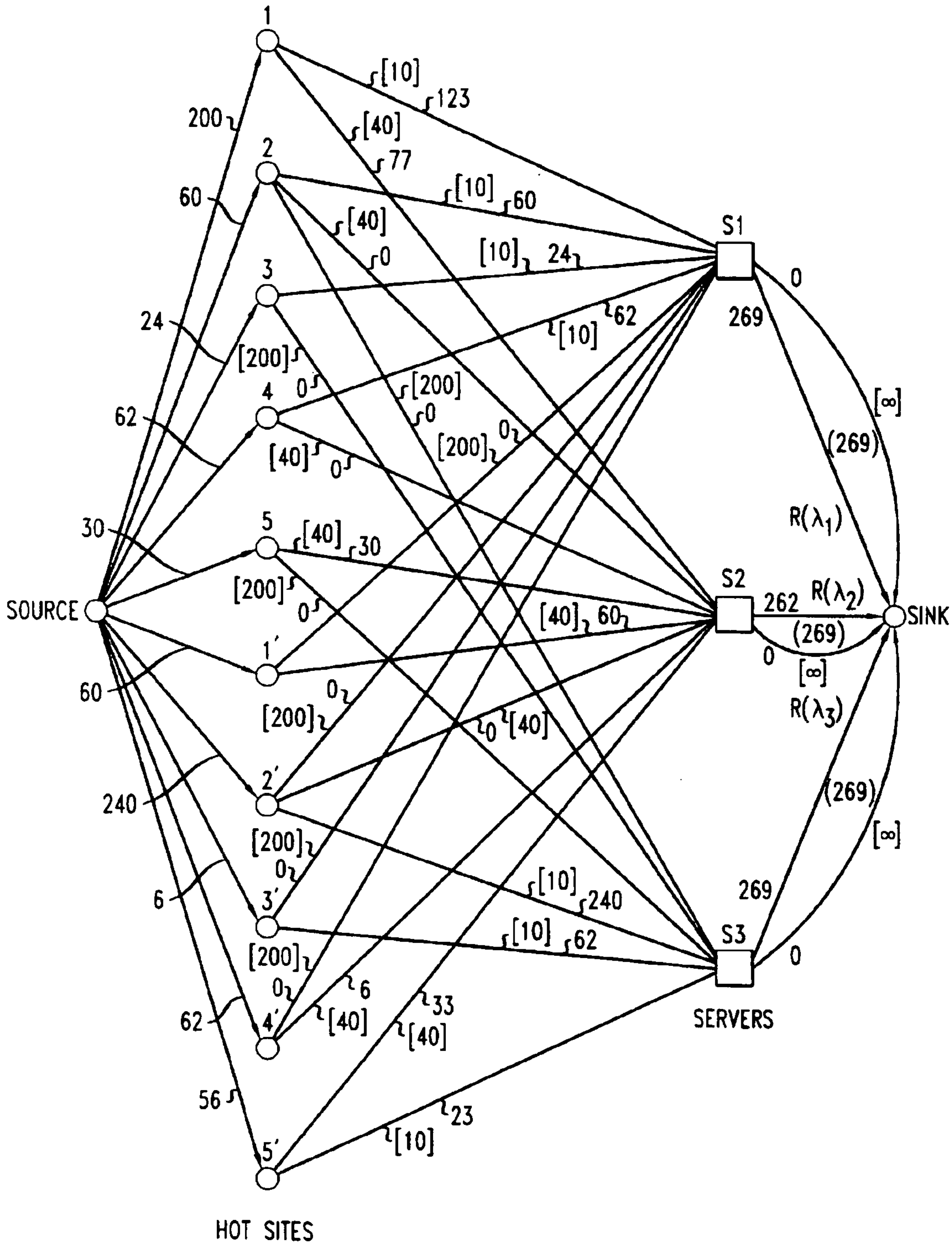
1
2
3
4
5
1′
2′
3′
4′
5′
SOURCE
DOCUMENTS
S1
S2
S3
SERVERS
SINK
0.25
0.1
0.04
0.02
0.05
0.1
0.4
0.01
0.02
0.01
[1] 0.17
[2] 0.08
[1] 0.1
[2] 0
[1] 0.04
[1] 0.02
[3] 0
[3] 0
[2] 0
[3] 0
[2] 0.05
[3] 0
[2] 0.1
0.07
[2]
[3] 0
[3] 0
[1] 0.33
[1] 0.01
[3] 0
0.02
[2]
0.01
[2]
[1]
[1] 0
[4]
(0.33)
0.33
(0.33) 0.33
[4]
0.34
(0.34)
[4]

FIG. 4



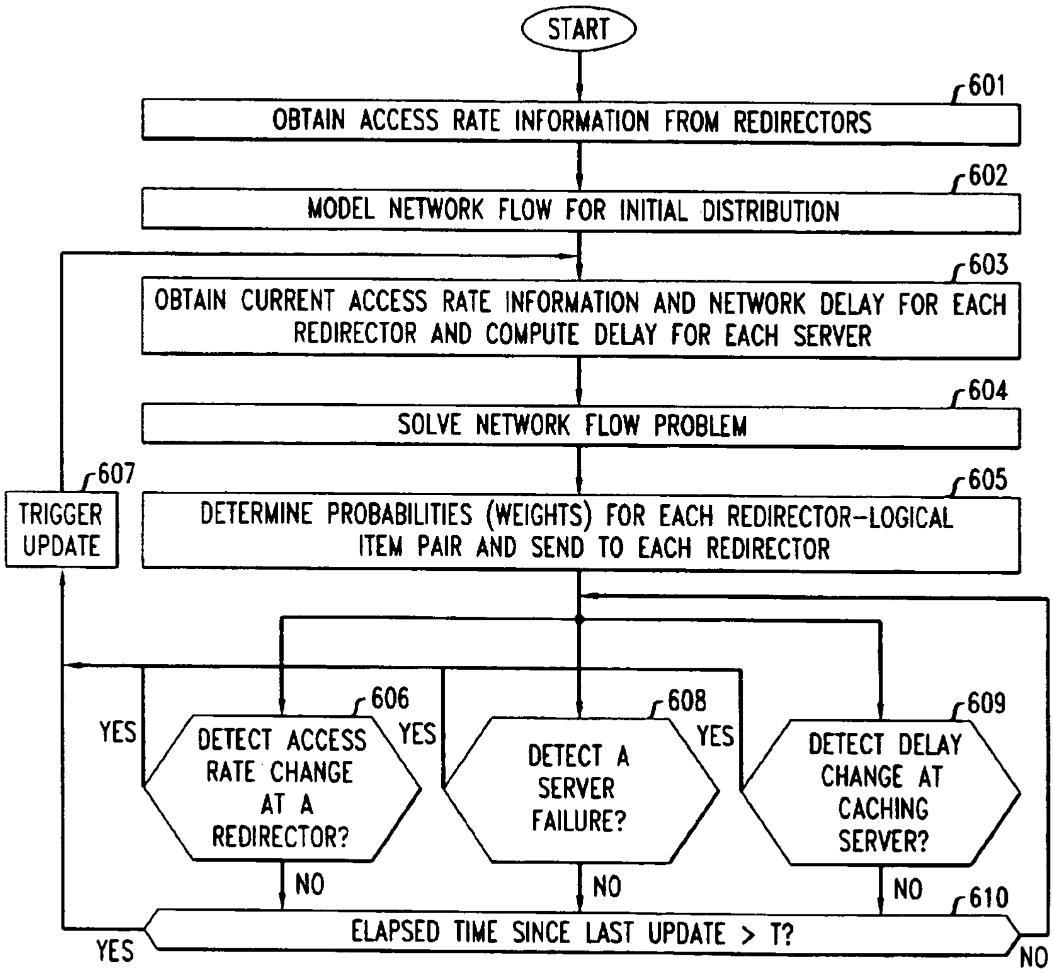
0.0006
0.0005
0.0004
P_k
0.0003
0.0002
0.0001
0
0.8
0.81
0.82
0.83
0.84
0.85
0.86
0.87
0.88
0.89
0.9
r

FIG. 5



1
2
3
4
5
1'
2'
3'
4'
5'
SOURCE
HOT SITES
S1
S2
S3
SERVERS
SINK
200
60
24
62
30
60
240
6
62
56
[10]
123
[40]
77
[10]
60
[40]
0
[10]
24
[10]
62
[200]
0
[200]
0
[40]
0
[200]
0
[40]
30
[200]
0
[40]
60
[200]
0
0
[40]
[200]
0
[10]
240
[200]
0
[10]
62
6
[40]
33
[40]
23
[10]
0
269
[∞]
(269)
R(λ1)
262
R(λ2)
(269)
0
[∞]
R(λ3)
(269)
[∞]
269
0

FIG. 6


START
601
OBTAIN ACCESS RATE INFORMATION FROM REDIRECTORS
602
MODEL NETWORK FLOW FOR INITIAL DISTRIBUTION
603
OBTAIN CURRENT ACCESS RATE INFORMATION AND NETWORK DELAY FOR EACH REDIRECTOR AND COMPUTE DELAY FOR EACH SERVER
604
SOLVE NETWORK FLOW PROBLEM
605
DETERMINE PROBABILITIES (WEIGHTS) FOR EACH REDIRECTOR-LOGICAL ITEM PAIR AND SEND TO EACH REDIRECTOR
607
TRIGGER UPDATE
606
DETECT ACCESS RATE CHANGE AT A REDIRECTOR?
608
DETECT A SERVER FAILURE?
609
DETECT DELAY CHANGE AT CACHING SERVER?
YES
YES
YES
NO
NO
NO
610
ELAPSED TIME SINCE LAST UPDATE > T?
YES
NO

METHOD AND APPARATUS FOR LOAD SHARING ON A WIDE AREA NETWORK

TECHNICAL FIELD

This invention relates to wide area networks such as the Internet, and more particularly, to a method and apparatus for minimizing the average delay per unit of time of all client requests incurred over all connections established for accessing server devices, such as web servers and proxy cache servers, which are located across the wide area network.

BACKGROUND OF THE INVENTION

In co-pending patent application Ser. No. 08/953577 entitled "Data Distribution Techniques for Load-Balanced Fault-Tolerant Web Access", by B. Narendran, S. Rangarajan (co-inventor herein) and S. Yajnik, assigned to the assignee of the present application, and which is incorporated herein by reference, a methodology is described for balancing the load on a set of devices connected on a wide area network such as the Internet. Specifically, the methodology for load balancing described in that application provides, in a first phase, an algorithm for distributing web documents, or objects, onto different servers such that the total access rates to each server (equal to the total number of connection requests that a server handles per time unit) are balanced across all the servers. Further, in a second phase of the methodology, a network flow-based algorithm is used to re-balance the access rates to each server in response to system changes without moving objects between the different servers.

In further detail, in the first phase of the load balancing methodology, logical items, such as the web documents, or objects, are mapped to different physical devices such as web servers, cache servers, ftp servers, etc., based on the a priori access rates that are known for requests from/to these web documents. This mapping, referred to the initial distribution, takes as an input the access rates of each web document, the number of replicas of these web documents that need to be made on the physical devices, such as document servers or caches, and the capacity of each of the physical devices, and produces a mapping between the web documents and the physical devices. It also produces as an output the probabilities (or weights) that will then be used by a redirection server to redirect requests from/to replicated web documents to one of the physical devices to which they are mapped. This initial distribution mapping is performed such that the load is balanced among the physical devices, or, i.e., the sum of access rates of requests to the web documents redirected to each physical device is balanced across all the devices. Load balance is achieved across the physical devices irrespective of the web documents that they handle.

In the second phase of the methodology, once the initial distribution of the web documents is performed, any change in the system parameters that affects the load balance is handled using a network flow load balance algorithm to determine new probabilities (or weights) with which the redirection server will then thereafter redirect requests from/to web documents to one of the physical devices to which they are mapped. Thus, instead of re-mapping web documents to different documents servers or caches to handle a perturbation in load, the load is re-balanced by changing the probability with which requests to each replicated web document is redirected to one of the plurality of physical devices to which that physical item is mapped. Examples of parameters that may change in the system include the load on each physical device and the capacity of each of the physical devices, the latter of which can instantly become zero upon the failure of a device.

The goal of load balancing, as described, is to balance across all physical devices the sum of the access rates of requests to the web documents redirected to each physical device. The latency, or delay, incurred in providing a response from a physical device to a request for a web document made by a client has not been previously considered.

SUMMARY OF THE INVENTION

In accordance with the present invention, minimization of request latency on a wide area network such as the Internet is the goal rather than pure load balancing. The load sharing methodology of the present invention minimizes delay by determining the probabilities, or weights with which web requests are redirected over the wide area network to the web servers so as to minimize the average delay of all connections across all servers per unit of time. Such redirection to the different servers is effected as a function of a logical item, the logical item being the factor that the redirector uses in determining where and with what weights the request is to be directed. In determining a solution to a non-linear programming optimization problem, the network delay associated with accessing a server and the server delay, which itself is a function of the access rate on that server, are taken into account. After an initial distribution of logical items is completed, such as for load balancing purposes in accordance with the aforedescribed prior art methodology, or another method, the following are determined: (1) the access rates, which are equal to the number of requests per unit time associated with each redirector-logical item pair; (2) the network delay, which is equal to the sum of the propagation and the transmission delays between the client and the server; and (3) the server delays incurred in processing a web request. Once these parameters are measured or mathematically computed they are used to determine the solution of a non-linear program optimization problem. This non-linear programming problem is formulated and solved as a minimum cost network flow problem to ultimately determine the probability or distributions, or weights, with which each redirector in the network will then redirect requests to the different servers which can satisfy them.

In a specific embodiment of the present invention, highly popular, a/k/a hot sites, are mapped to particular caching servers dispersed in a wide area network, with each hot site being mapped to one or more caching servers. Statistics of access rates to each hot site are dynamically determined by each redirector in the network from the traffic flow and reported to a central management station (CMS). Network delay is similarly measured by each redirector and reported to the CMS and server delay at each server is computed using a queuing model of each server. Using these parameters, the CMS solves a network flow problem to determine the weights with which each redirector will then probabilistically forward requests for a hot site to the different plural servers which are responsible for that requested site. As the access rate statistics, as well as possibly the measured network delay and server delay, dynamically change, the CMS, using the network flow algorithm, recalculates the weights and forwards the adjusted weights back to each redirector. The weights with which each redirector forwards requests for specific documents to a particular server re therefore continually modified

in a manner that minimizes the average delay based on the most recent access rate, network delay and server delay statistics.

BRIEF DESCRIPTION OF THE DRAWING

FIG. 1 is a block diagram of a system on a wide area network showing two redirectors which direct requests from groups of clients for hot sites to plural caching servers which are responsible for such hot sites in accordance with a mapped relationship;

FIG. 2 is a prior art network flow solution for load balancing for a system containing a single redirector with plural caching servers;

FIG. 3 is a network flow solution for load balancing for a system containing two redirectors;

FIG. 4 is a graph showing the probability of a server dropping a connection request;

FIG. 5 is a network flow solution of a load sharing problem for the system in FIG. 1; and

FIG. 6 is a flowchart detailing the steps of the present invention.

DETAILED DESCRIPTION

The present invention is illustrated below in conjunction with exemplary client/server connection s established over the Internet using the Transmission Control Protocol/Internet Protocol (TCP/IP) standard. It should be understood, however, that the invention is not limited to use with any particular type of network or network communication protocol. The disclosed techniques are suitable for use with a wide variety of other networks and protocols. The term "web" as used herein is intended to include the World Wide Web, other portions of the Internet, or other types of communication networks. The term "client request" refers to any communication from a client which includes a request for information from a server. A given request may include multiple packets or only a single packet, depending on the nature of the request. The term "document" as used herein is intended to include web pages, portions of web pages, computer files, or other type of data including audio, video and image data.

FIG. 1 shows an exemplary web server system **100** in accordance with an illustrative embodiment of the invention. The system includes a first redirection server R1 (**101**) local to local area network **102** and a second redirection server R2 (**103**) local to local area network **104**. Local area networks **102** and **104** are separated and connected over a wide area network (Internet **105**). A plurality of clients, **106-1–106**-N, are connected to local area network **102** and a plurality of clients, **107-1–107**-M, are connected to local area network **104**. All requests for web documents made by clients **106-1–106**-N are passed through their local redirection server **101** and redirected to a caching server on system **100**. Similarly, all requests for web documents made by clients **107-1–107**-M are passed through their local redirection server **103** and redirected to a caching server on system **100**. In the illustrative embodiment, a first caching server S1 (**110**) is connected local to local area network **102**, a second caching server S2 (**111**) is connected to the wide area network Internet **105**, and a third caching server S3 (**112**) is connected to local area network **104**. In addition to caching individual web documents, each of these caching servers is responsible for particular hot sites, or web sites to which a large number of client requests are directed. Each caching server is responsible for one or more of such hot sites and each such hot site may be associated with more than one of the caching servers.

In the system **100**, communication between clients and caching servers is effected over TCP/IP connections established over a network in a conventional manner. Each of the elements of system **100** may include a processor and a memory. The system **100** is suitable for implementing Hypertext Transfer Protocol (HTTP)-based network services on the Internet in a manner well known to one skilled in the art. For example, a client may generate an HTTP request for a particular web document on a hot site by designating a uniform resource locator (URL) of that document at the domain name of the hot site. Such a request is passed through the requesting client's local redirection server and redirected to one of the caching servers, **S1**, **S2** or **S3**, which is responsible for that hot site. A TCP/IP connection is then established between the requesting client and the particular caching server responsible for that hot site as selected by the local redirection server. If the particular requested document is available at that caching server, it is supplied to the requesting client. If not, a separate TCP/IP connection is established by the caching server to the actual hot site from where a copy of the requested web document is obtained and forwarded to the client.

In the aforenoted co-pending patent application, a load distribution algorithm is presented for determining an initial distribution of a set of documents across servers and a determination of the probabilities, or weights, with which a redirector server should redirect requests to those particular servers that contain a replica of a requested document. Such load distribution is undertaken to balance the load on the set of servers where the definition of load balance is that the sum of access rates of requests to the documents redirected to each of the servers containing the documents is balanced across all the servers. Using as input the access rates of each document, the number of replicas of these documents that need to be made on the servers, and the capacity of each server, an initial distribution algorithm, described in the aforenoted co-pending application incorporated herein, produces a mapping between the documents and the servers as well as producing as output the probabilities (or weights) to be used by a redirecting server to redirect requests from/to replicated documents to one of the severs to which they are mapped to achieve the desired load balance.

FIG. 2 shows a flow network model in the co-pending patent application between a source and a sink for five documents, numbered 1–5 distributed on three servers, S1, S2 and S3, which each have equal scaled capacities of 0.333. The flow network is of a type described in, for example, R. K. Ahuja et al., "Network Flows: Theory, Algorithms and Applications", Prentice Hall, 1993, which is incorporated by reference herein. The scaled access rates to each of the documents, totaling 1, are shown on the five arcs between the source node and the redirector/document nodes. These access rates represent the probabilistic proportion of requests arriving at the redirector for each of the five documents. As can be noted from the arcs between the redirector/document nodes and the three server nodes, server S1 has stored replicas of document **1**, **2**, **3** and **4**, server S2 has stored replicas of documents **1**, **2**, **4** and **5**, and server S3 has stored replicas of documents **2**, **3** and **5**. These documents have been distributed in accordance with the initial distribution algorithm described in the aforenoted co-pending application to assure that at least two replicas of each document are stored in the set of three servers. The numbers on the arcs between the redirector/document nodes and the servers nodes represent the network flow solution for

dividing the incoming access rate to the redirector/document nodes for each document to the servers that contain the document so that the total load to the servers S1, S2 and S3 is balanced. In the FIG. 2, the redundant arcs from the servers to the sink represent arcs that have infinite capacity but have a "high" cost associated with them. The cost on all the other arcs is equal to zero. The "high" cost arcs are used for overflows that may result when a change occurs in the system.

The distribution represented in FIG. 2 is a maximum-flow minimum-cost solution to the corresponding network flow problem that models the load balancing problem which must be solved to determine the desired solution. The solution is obtained using the mathematical language AMPL as described by R. Fourer et al., "AMPL: A Modeling Language for Mathematical Programming," The Scientific Press, 1993, which is incorporated by reference herein, in conjunction with a non-linear program solver such as MINOS. The numbers on the arcs between the redirector/document nodes and the server nodes represent the portions of the access rates for each document that are redirected to each server that contains the document. Thus, for example 0.175 of the 0.35 access rate requests for document 1 is redirected to server S1, and a similar portion is redirected to server S2. Thus, requests for document 1 received by the redirector should be redirected to servers S1 and S2 with equal weights or probabilities. Similarly, 0.113 of the 0.5 access rate for document 2 should be redirected to server S1, 0.108 of the 0.5 access rate should be redirected to server S2, and 0.279 of the 0.5 access rate should be redirected to server S3.

The corresponding weights, or probabilities, for redirection of a request for document 2 are thus 0.226, 0.216 and 0.558 for servers S1, S2 and S3, respectively.

Although the above example and others in the co-pending patent application illustrate load balancing initial distributions involving a single redirector server which redirects requests to a plurality of document servers on which documents are replicated, the single redirector model can be extended to accommodate multiple redirectors. FIG. 3 models the same web server system as in FIG. 2, but includes two redirectors, as in the system of FIG. 1. In FIG. 3 redirector/document nodes 1 through 5 now represent requests for documents 1 through 5, respectively, which are redirected to the servers S1, S2 and S3 by redirector 1, and redirector/document nodes 1' through 5' represent requests for documents 1 through 5, respectively, which are redirected to the servers S1, S2 and S3 by redirector 2. The scaled access rate of each document is now, however, divided between the two redirectors. As an example, the scaled access rate for document 1, which in FIG. 2 is 0.35, is divided between the two redirectors so that the access rate to document 1 redirected through redirector 1 is 0.25 and the access rate redirected through redirector 2 is 0.1. The access rates for document 2 through redirectors 1 and 2 are 0.1 and 0.4, respectively; for document 3 through redirectors 1 and 2 are 0.04 and 0.01, respectively; for document 4 through redirectors 1 and 2 they are each 0.02; and for document 5 through redirects 1 and 2 they are 0.05 and 0.01, respectively. In the single redirector model of FIG. 2, equal proximity between the redirector and the servers was assumed so that costs on the arcs between the document nodes and the server nodes were not considered as variable parameters. It can be assumed, however, as shown in the network of FIG. 1, that the distances, $d1j$, $1 \leq j \leq 3$, between redirector 1 and servers S1, S2 and S3 are such that $d11<d12<d13$. Thus, as in the web network in FIG. 1, S1 is local to redirector 1 on local network 102, is separated from server S2 on the Internet 105 by a mid-range distance, and is furthest from S3 which is connected on a distant local network 104. Thus, for the two-redirector arrangement in FIG. 3, redirector 1 should redirect requests for a document to server S1 if it is available there and redirect requests for a document to server S2 and then S3 only if the capacity of S1 is exceeded. To model this distance factor, costs are assigned to the arcs between the redirector/document nodes and the server nodes according to the distance between the redirector and the server. For example, requests for document 2 from redirector 1, if redirected to nearby server S1 are designated as having a cost per unit flow of 1, if redirected to mid-distance server S2 are designated as having a cost per unit flow of 2, and if redirected to distant server S3 are designated as having a cost per unit flow of 3. These same costs are assigned between the other redirector/document nodes associated with redirector 1 and the three servers. In a similar manner, redirector 2 is considered local to server S3, as shown in FIG. 1, is separated from S2 by a mid-range distance, and is separated from S1 by a long distance. In FIG. 3, the numerals in the square parentheses represent the costs on the arcs between the redirector/document nodes and the server nodes that contain replicas of the documents.

The cost on the overflow arcs from the servers to the sink needs to be carefully controlled. If load balance is the primary objective, then the costs on these arcs are chosen such that they are larger than the largest cost on the arcs that connect the redirector/document nodes to the server nodes. Otherwise, the network solution will lead to redirecting a flow to a close proximity device even if the load balance condition is violated as opposed to redirecting the flow to a device that is further away without violating the load balance requirement. For example, if the cost on the overflow arc from node S1 to the sink is chosen to be between 2 and 3, for all documents that are available in S1 and S3, redirector 1 will deterministically send the request to S1 possibly overflowing the capacity of server S1 even if server S3 has spare capacity. If the cost on the overflow arc is less than 2, requests for documents available on S1 and elsewhere will always be sent to S1 even if it overflows the capacity of Si. Where load balance is the primary objective, therefore, the costs on the overflow arcs are chosen to be 4, which is larger than the cost on any of the arcs between the redirector/document nodes and the server nodes. FIG. 3 shows the network flow solution, with flows between redirector/document nodes being specified without any parentheses around them, capacities specified within curved parentheses, and, as noted, costs specified within square parentheses. Where the capacity or the cost of an arc is not specified, its default value is 1. Using the determined flow values, the redirection probabilities, or weights, with which a request is forwarded to a particular server that has a stored copy of the requested document can be determined. Thus, for example, the flow on the arc from the redirector 1/document 1 node to S1 specifies that of the 0.25 units of scaled flow for the document 1/redirector 1 pair, 0.17 units should be redirected to server S1.

Thus, if a request for document 1 arrives at redirector 1, with a probability of 0.17/0.25, it will be directed to server S1 and with probability 0.08/0.25 it will be directed to server S2.

Unlike the solutions for load balancing focused on by the prior art, the present invention provides a solution for load sharing which considers the network delay associated with accessing a server and the server delay, which itself is a function of the access rate on that server. A network flow

approach to this problem is considered after an initial distribution is completed for load balance. The aim of this load sharing solution is to minimize the average delay of all connections across all servers per unit of time. FIG. 1 is the network model of a preferred embodiment of this invention. As previously noted, hot sites rather than individual documents are mapped to servers S1, S2 and S3, which, in this embodiment, are proxy cache devices. The network delay, which is the sum of the round-trip propagation and the transmission delays, is modeled by specifying them in the network flow model as costs on the arcs from the redirector/logical item nodes to the proxy cache server nodes, where the logical items are the different cached hot sites. The links from the proxy cache server nodes to the sink have a cost associated with them which is a function of the flow through these links. This cost models the device delay. The larger the number of connections a server serves per unit of time, the larger the device delay for each of the connection requests. The capacities of these links are not now used to specify a balance of load among the server devices, but to specify a limit on the average number of connections per time unit sent to the server device above which the connections may be dropped. As is described below, the proxy cache server delay and the capacity of the proxy cache server are calculated using a queuing model.

The caching server is modeled as an M/M/1/K queue to which connection requests arrive with a Poisson arrival rate. The service time at the device is exponentially distributed. There is one server (CPU) and the maximum queue space at the server is K. This maximum queue space specifies the maximum number of simultaneous connections that the proxy cache server can support. If the number of simultaneous connection requests is greater than K, then requests will be dropped. If it is assumed that λ and μ are the expected arrival and service rates, then it can be shown that the probability that there are k jobs in the queuing system is given by:

$$P_k = \frac{1-\frac{\lambda}{\mu}}{1-\left(\frac{\lambda}{\mu}\right)^{K+1}}\left(\frac{\lambda}{\mu}\right)^k \quad 0 \le k \le K \tag{1}$$

Then, the average queue length at the server is given by

$$\overline{N} = \sum_{k=0}^{K} kP_k.$$

Using Little's Law (see L. Kleinrock, *Queuing Systems*, Vol. 1, Prentice Hall), the average response time or the device delay at the server can be computed as

$$R = \frac{\overline{N}}{\lambda}.$$

This device delay represents the sum of the queuing delay and the service delay at the server.

By letting

$$\rho = \frac{\lambda}{\mu}$$

and using the above equations and after manipulations, it can be shown that a closed form solution for R is given by:

$$\frac{1}{\lambda}\left(\frac{\rho}{1-\rho}\right)\left[1-\frac{(K+1)\rho^K(1-\rho)}{1-\rho^{K+1}}\right] \tag{2}$$

It can be noted that when $K \to \infty$, the response time approaches

$$\frac{1}{\lambda}\left(\frac{\rho}{1-\rho}\right)$$

which is the response time of a M/M/1 system with infinite queue size.

Server capacity is determined by the allowed dropping probability of connection requests. When a connection request arrives at a server, the request is dropped if there already are K connections being served by the server. The probability that an arriving request sees K existing connections at the server is given by P_K. If it is assumed that the dropping probability is to be less than P_d, then the maximum flow F (and thus the capacity) that is allowed at a server is calculated as $P_K<P_d$. From the equation for P_K, it is found that

$$P_K = \frac{\rho^K(1-\rho)}{1-\rho^{K+1}}.$$

F is substituted for λ in P_K and this valued is bounded to be within P_d. Solving for F then provides the maximum flow that can be sent to a server. From the following equation, F can be computed:

$$\frac{\left(\frac{F}{\mu}\right)^K\left(1-\frac{F}{\mu}\right)}{1-\left(\frac{F}{\mu}\right)^{K+1}} < P_d \tag{3}$$

The load sharing algorithm is illustrated using an example applied to the proxy caching hot site network arrangement of FIG. 1. The queuing module previously discussed is used to compute the server delay at each proxy caching server. This server delay will be a function of the amount of flow sent to that server (which represents the number of connection requests per time unit sent to that server). The maximum flow that can be sent to a server (which represents the maximum rate at which connection requests are redirected to a server) such that the specified condition on the connection dropping probability is satisfied is also computed.

In this example, the maximum number of simultaneous connections that can be handled by a proxy caching server (K) is set to 50. This means that if the proxy caching server is handling HTTP requests, it can handle 50 such requests simultaneously. It drops extra connection requests that come in if it is already handling 50 requests. The maximum connection loss probability, P_d, is set to be 0.0001, which means that on an average only 1 in 10000 connections is not completed and is dropped at the server. FIG. 4 is a plot of the connection loss probability P_K versus $r=F/\mu$ with K=50. From the plot it can be seen that for $r \leqq 0.865$, $P_K<0.0001$. That is, for $F<0.865\times\mu$, $P_K<P_d$. Depending on the service rate μ at a server, the maximum flow F that can be sent to that server can be computed. The service rate μ at a server specifies the rate at which HTTP connection requests are satisfied. Assuming that on an average each object is 4K bytes and that proxy servers are connected through a 10

Mbit/sec ethernet, it can be reasonably assumed that the bottleneck is the bandwidth of the network and not the CPU performance. Then, to transmit a 4K byte object over a 10 Mbit/sec network requires 3.2×10^{-3} seconds, which means that the service rate μ is

$$\frac{1}{3.2\times10^{-3}} \doteq 312.$$

Thus, the proxy cache server can handle approximately 312 HTTP connection requests per second. If it can be assumed for this example that all servers can handle approximately 312 connections per second, then F should be found by $0.865\times312 \approx 269$ at all servers. The server delay as a function of the flow λ sent to the cache is given by Equation (1) above. Given the above parameters, the server delay $R(\lambda)$ is given by:

$$\frac{1}{\lambda}\left(\frac{\left(\frac{\lambda}{312}\right)}{1-\left(\frac{\lambda}{312}\right)}\right)\left[1-\frac{(51)\left(\frac{\lambda}{312}\right)^{50}\left(1-\left(\frac{\lambda}{312}\right)\right)}{1-\left(\frac{\lambda}{312}\right)^{51}}\right] \tag{4}$$

For purposes of the example, it is assumed that the network delay from redirector **101** to nearby caching server S1 on the same local network is 10 ms, to mid-distance caching server S2 on the Internet **105** is 40 ms, and to caching server S3 on the distant local network **104** is 200 ms. Similarly, the network delay from redirector **103** to nearby caching server S3 is assumed to be 10 ms, to mid-distance caching server S2, 40 ms, and to distant caching server S1, 200 ms.

FIG. **5** illustrates a network flow model **500** of the server system **100** in FIG. **1** in which the parameter values noted above have been incorporated. From the initial distribution, caching server S1 is responsible for hot sites **1**, **2**, **3** and **5**; caching server S2 is responsible for hot sites **1**, **3**, **4** and **5**; and caching server S3 is responsible for hot sites **2**, **3** and **5**. This mapping is noted in FIG. **1** within the S1, S2 and S3 caching servers. This is parallel to the mapping of documents on the three servers used in conjunction with the description of FIG. **2** for load balancing. From source **501**, a total of 800 connection requests per second are generated for all hot sites. The requests flowing through redirector **101** are represented by the arcs between the source **501** and the redirector/hot site nodes **1**–**5** and the requests flowing through redirector **103** are represented by the arcs between the source **501** and the redirector/hot site nodes **1'**–**5'**. The numbers associated with the arcs from source **501** to the redirector/hot site nodes **1**–**5**, and **1'**–**5'** specify the connection requests flowing through each respective redirector to each hot site. The cost representing the network delays are shown in the square brackets on the arcs between each redirector/hot site node and the server nodes. Each caching server is assumed, as noted above, to support up to 50 connections per time unit without dropping any connections. If more requests are directed to these devices, then some connections may be dropped. All overflow connections are handled through the overflow arcs between the server nodes and the sink **502**. The cost parameter chosen for these arcs (represented in FIG. **5** as infinity [∞]) should be chosen to be a larger number than the cost on all other arcs. The server delay at the proxy caches, shown as the cost on the arcs between each proxy cache server and the sink **502**, is given by $R(\lambda_1)$, $R(\lambda_2)$, and $R(\lambda_3)$, where λ_i is equal to the connection requests directed to caching server S1 and $R(\lambda_i)$ is calculated using equation (4).

The model in FIG. **5** is solved using the aforenoted AMPL mathematical programming language and the MINOS non-linear program solver with a goal of finding a solution to the non-linear problem such that the average response time for all of the 800 connections arriving per second is minimized. AMPL is a mathematical programming language that can be used to specify different optimization problems. It is used here to specify a minimum cost network flow optimization problem. For example, the load sharing network flow model of FIG. **5** can be modeled using the program shown in Appendix 1. The program defines the nodes in lines **101**–**103**, the connectivity of the node in lines **104**–**106**, specifies that the flow in must equal the flow out at line **107**, specifies various criteria line **108**–**113**, the function to be minimized in line **114** (postulated in the program as a minimization of total cost), and the different constraints of the problem in lines **115**–**118**. The different parameters of this network, such as the number of hot sites, the number of redirectors and the number of proxy cache servers can be varied by specifying them in a data file with which the program is associated. Further, the input parameters of the model such as access rates, delays and capacities are also specified in this data file. The data file for the example in FIG. **5** is shown in Appendix 2. The AMPL environment takes the model file and data file and uses one of several different solvers to solve the optimization problem depending on the nature of the problem. As the specified problem is a non-linear optimization problem, the MINOS solver is used to solve the problem. The output of the solver provides the flow on each link between the redirector/hot-site nodes to the server. These flow values, noted on these arcs are then used by the redirectors to determine the probability or weights with which a request for a hot site is redirected by that redirector to a particular proxy cache server that is responsible for that requested hot site.

With reference to the mathematical example shown in FIG. **5**, of the 200 requests per second that arrive for hot site 1 at redirector 1, the solution indicates that 123 requests are redirected to caching server S1 and **77** requests are redirected to caching server S2. This means that when a subsequent request for hot site **1** arrives at redirector **1**, it should be redirected to server S1 with a probability of 123/200 and to S2 with a probability of 77/200. Similarly, when a subsequent request for hot site **5** arrives at redirector **2**, it is redirected to server S2 with a probability of 33/56 and to server S3 with probability 23/56. The solution illustrated in FIG. **5** results in all the other requests arriving at a redirector being redirected to the closest server. All the requests that do arrive at server S2 in the example are those that cannot be served at a closer server because a) the request hot site is not cached at the closer server; or b) redirecting the request to the closer server will violate its capacity requirement. The flow, the λ values, going into each server are shown on the arcs from the server nodes to the sink node **502**. The capacity of each such node is shown on these same arcs in the curved parentheses. As can be noted, servers S1 and S3 are filled to their capacity while server S2 still has some spare capacity available.

It can be noted from FIG. **5** that the load is now not balanced but shared among the servers such that the average delay is minimized. From the solution, the total minimized delay of all the 800 connections is calculated to be 15860 ms for an average delay of 19.825 ms per connection. The number of connections redirected to each device is noted to be less than or equal to the maximum number of connections that can be handled and thus the condition on the probability of dropped connections is satisfied. Thus, as can be noted in FIG. **5**, there is zero flow in the overflow arcs between the server nodes and sink **502**.

In accordance with the embodiment of the present invention in FIG. 1, a connection management station (CMS) **115** performs a network flow computation to calculate the weights with which redirectors **101** and **103** will redirect further incoming requests for one of the five numbered hot sites. Once the calculation is performed, the resultant weight values are sent back to the appropriate redirector over, for example, a TCP/IP connection. CMS **115** is shown connected to the Internet **105** but in actuality can be connected anywhere on the network in FIG. 1, such as on local network **102** or on local network **104**. In order for the CMS **115** to perform a network flow calculation it periodically collects access rate and network delay information from redirectors **101** and **103** and server delay information from the caching proxy servers S1, S2 and S3.

Access rate information to each hot site is determined by each redirector by associating the destination address in the SYN packets with a set of hot site IP addresses. Alternatively, this information can be collected by examining, at the redirectors, the HOST field in the GET packets.

In the network configuration of FIG. 1 in which redirector **101** is local to clients **106-1–106**-N and redirector **103** is local to clients **107-1–107**-M, the redirectors do not have to treat traffic from different local clients in different manners. Network delay can therefore be accounted for by just considering the delay from a each redirector to each of the different caching servers. For a symmetric flow of traffic in which packets from the clients to the caching servers and from the caching servers to the clients flow through the redirector, network delay can be tracked by each redirector by computing the time between redirecting a SYN packet to a particular caching server and receiving the corresponding SYN ACK packet back from that server. In an asymmetric flow of traffic in which the client-to-server traffic flows through the redirector but the reverse traffic flows directly from the server to the client, the SYN ACK packet will not flow through the redirector. Therefore, network delay can be measured with another mechanism such as by PINGing the servers periodically.

Server delay at each of the caching proxy servers S1, S2 and S3 is calculated using the aforedescribed queuing model which resulted in equation (4) from which the server delay $R(\lambda_i)$ is determined as a function of the flow λ_i into server S1.

Once an initial distribution of hot sites onto caching servers is performed based on access rate information to achieve load balancing in the manner specified in the co-pending application, CMS **115** performs a network flow computation for purposes of load sharing to determine the weights with which the redirectors should redirect requests to replicated hot site caches. Using these determined weights at each redirector minimizes the average delay of all connections across all of the caching servers per unit of time. This load sharing network flow computation is continually updated by periodically collecting current access rate and network delay information from each redirector and server delay information from the cache servers. Thus, the weights are continually updated based on the latest access rate, network delay and server delay information. Further, if the CMS **115** detects a failure of a caching server, it will trigger a network flow computation. In this case, the corresponding node is removed from the network flow model as well as all arcs incident upon it. Further, the service rate, μ, may change at a caching server if, for example, one of two processors fails or if a device is replaced by another device with a higher performance CPU. This will affect two parameters: a) the delay at a server given that a specific number of connections are redirected to that server; and b) the capacity of the server in terms of the maximum flow that can be redirected to that server. As a result of a service rate change, a network flow computation can be triggered. Even further, if the access rate to a particular hot site suddenly and dramatically changes, a redirector will trigger a network flow computation. In this case, the flows on the arcs from the source to the redirector/hot site nodes are changed on the network flow model. Changes in other parameters can also affect the network flow computation. Thus, if the relative round-trip delay from the redirectors to the caching servers changes, the costs associated with the arcs from the nodes representing the redirectors to the server nodes are changed. Also, since server load and server delay are determined by the number of requests redirected to a caching server, a change in such number of requests per second redirected to a server will change the server delay parameter.

FIG. **6** is a flowchart detailing the steps of the method of the present invention. At step **601**, access rate information is obtained by CMS **115** from each redirector. At step **602**, using this access rate information, an initial distribution of hot sites on the caching servers is determined using the prior art load balancing network flow algorithm. Once the initial distribution is determined, at step **603**, CMS **115** obtains current access rate and network delay information for each redirector, and the server delay of each server is calculated. Using these inputs, at step **604**, a network flow problem for load sharing is solved. At step **605**, the probabilities (or weights) for each redirector for each hot site pair are determined and sent to each redirector. At decision step **606**, a determination is made whether there has been an access rate change at a redirector. If yes, an update is triggered at step **607**, which in turn causes the current access rate, network delay, and server delay to be determined or calculated back at step **603**. Similarly, at decision step **608**, a determination is made whether a server failure is detected. If yes, an update is triggered again at step **607**. Further, at decision step **609**, a determination is made whether or not a change in the delay at a caching server is detected. If yes, an update is triggered at step **607**. If an access rate change, a server failure, or a caching server delay change are not detected at either decision steps **606**, **608** or **609**, respectively, then, at decision step **610**, a determination is made whether the elapsed time since the last update has exceeded a threshold period of time, T. If not, the flow returns to the inputs of decision steps **606**, **608** and **609**. If the elapsed time has exceeded T, then an update is triggered at step **607**.

In the described embodiment, CMS **115** performs a centralized data gathering and network flow analysis function. Such functions could alternatively be performed at either redirector through TCP/IP communication between both such redirectors for exchanging access rate and delay information. Further, server delays need to be communicated to the redirectors.

Although described in conjunction with a system designed to achieve load balance across plural caching servers containing replicated hot sites for which in the network flow model the logical item at each redirector/logical item node represents a hot site, the present invention is not limited to such an arrangement. Thus, rather than having the logical items which are mapped to different servers being hot sites, as described above, the logical items could be any group of hot documents which are mapped onto a plurality of local caching servers in accordance with the document's origin server IP addresses. The origin server IP

addresses of the documents are hashed to different groups. The different groups are then considered the logical items which are mapped onto the plural caching servers. Since many origin servers use multiple IP addresses, to avoid the same origin server name from being mapped to multiple caches, the hashing function is chosen so that the all IP addresses for an origin server are mapped to the same group out of a possible 256 different groups. Since origin servers normally use a contiguous block of IP addresses, then if the hashing is based on the first 8 bits of the origin server IP address, this contiguous block of IP address will be automatically mapped to the same group.

Alternatively, the logical items could be any group of hot documents identified by the URLs, as might be done if a content-smart switch is used as the redirector. For this case, documents may be grouped according to the type of objects requests, such as .html, .jif, .jpeg, etc.

Another way to perform the group mapping could be based on the logical names of the origin servers. Different logical names could be mapped to different groups. This requires looking at the HOST field found in the GET packets to keep track of the access rates to the different servers, and thereby to the groups. The initial distribution algorithm can then be used to decide where the logical items should be distributed. Alternatively, the initial distribution can be performed based on other information such as the proximity of the cache to the clients requesting a specific document. Once the initial distribution is performed by either these method, or by another method, the load sharing algorithm of the present invention is performed to minimize the average delay of all connections across all of the caching servers per unit of time. Thus, access rate information to each formed group, network delay and server delay are determined as inputs to the network flow problem. From the network flow solution to the non-linear optimization problem, the optimum weights for each redirector/group of documents for the desired load sharing are determined. These weights, then determine the probabilities with which the redirector directs a request for a document within one of the replicated logical items, the latter being one of the formed groups of origin server IP addresses.

In the embodiments discussed hereinabove, it has been assumed that the clients are local to one of the redirectors. Therefore, the network delay between the client and the redirector has not been considered as a factor in the solution to the load sharing problem. The other possibility is for the redirectors to be closer to a plurality of servers. In this scenario, a client's request for a logical name is resolved by a Domain Name Server into a redirector's IP address, which in turn redirects the request to a server in a cluster of essentially duplicated servers. In this case, server side load balancing is achieved, in accordance with the present invention, so as to minimize the average delay per unit of time of all requests of all connections across all the servers in the cluster. In such an embodiment, client IP addresses are mapped into groups by a simple hashing function. As an example, the first 8 bits of the client IP addresses can be used to determine which of 256 client groups a client is mapped into. Other hashing functions could be found that evenly distribute the clients among the groups such that the access rate of clients allocated to each of the groups are evenly distributed among the groups. Regardless of the hashing function, each group becomes a logical item which is mapped onto the physical devices, which are, in this case, the back-end servers. The initial distribution algorithm can be used to map these logical items to the servers. The network flow based algorithm can then be used to compute the redirector probabilities associated with each redirector/logical item pair, which in this case is a redirector/client group pair. In the network flow model then, the arcs between the source and each redirector/client group pair represent the requests generated from each group that need to be redirected to one of the back-end servers. The load sharing solution to the network flow problem, in accordance with the present invention, will produce the weights with which the redirectors direct requests from each of the client groups to the plural servers. The cost associated with each arc represents the delay. Thus, the cost on an arc between the source and one of the redirector/client group nodes represents the network delay encountered between a client group and the redirector, while the cost on an arc between a redirector/client group node and a server represents network delay between the redirector and one of the servers, plus that server's delay. In the hot site embodiment of the present invention previously described, it was assumed that the clients and the redirectors were local to one another so that the delay between each was a constant small value, and thus not a variable. With server side load sharing, this delay is certainly a variable that is considered in the network flow load sharing solution.

If the redirector is local to the back-end servers, then the network delay computation does not have to include the delay from the redirector to the servers. By grouping the clients together in groups according to their IP addresses, the clients within each group are likely to be geographically proximate. Thus, the network delay can be determined for each group of clients and a specific server by determining the network delay from each client group to the redirector. For a symmetric traffic flow, the delay from the client group to the redirector can be calculated at the redirector by keeping track of the time between redirecting a SYN ACK packet from the server to the client group and the time an ACK packet for the SYN ACK packet is received at the redirector as part of the three-way TCP handshake. With an asymmetric traffic flow model, the SYN ACK packet does not flow through the redirector. In this case, the redirector can periodically PING one or more clients in a client group to calculate the delay from that client group to the redirector. Further, the redirector separately keeps track of the access rates from each client group in order to solve the network flow for load sharing.

If the redirector, rather than being local to the back-end servers, functions as a virtual server which, upon receipt of a request, redirects the request to one of a plurality of servers located in different locations on the wide area network, then the delay between the redirector and each server also needs to be taken into account in addition to the delay between each client group and the redirector. If the traffic flow is symmetric, the redirector can calculate the network delay from a client group to a server by adding the delay from the client group to the redirector to the delay from the redirector to the server. The delay from the redirector to the server can be calculated as previously described by keeping track of the time between redirecting a SYN packet to a server and receiving the corresponding SYN ACK packet. As before, the delay from the client group to the redirector can be calculated at the redirector by keeping track of the time between redirecting the SYN ACK packet from the server to the client group and the time an ACK packet for this SYN ACK packet is received at the redirector. If the traffic flow is asymmetric, the delay on the traffic that flows directly from the server to the client is of interest. The client to server delay is not as an important parameter since most of the HTTP traffic flows from the servers to the clients. The

mechanism used for the symmetric flow case can be used as an approximation for the server-to-client delay or it can be estimated by the redirector using geographical information, such as that provided by IANA as used in Classless Inter-Domain Routing Protocol.

The network flow model can thus be used to solve the load sharing problem when the logical item in the redirector/logical item pair is associated with the clients making requests to a server or the servers providing responses to requests from clients. Further, the network model is flexible to provide a solution when a delay is associated with the link between a client group and the redirector, on the link between the redirector and the server, or on both links.

The foregoing therefore merely illustrates the principles of the invention. It will thus be appreciated that those skilled in the art will be able to devise various arrangements which, although not explicitly described or shown herein, embody the principles of the invention and are included within its spirit and scope. Furthermore, all examples and conditional language recited hereinabove are principally intended expressly to be only for pedagogical purposes to aid the reader in understanding the principles of the invention and the concepts contributed by the inventors to furthering the art, and are to be construed as being without limitation to such specifically recited examples and conditions. Moreover, all statements hereinabove reciting principles, aspects, and embodiments of the invention, as well as specific examples thereof, are intended to encompass both structural and functional equivalents thereof. Additionally, it is intended that such equivalents include both currently known equivalents as well as equivalents developed in the future, i.e., any elements developed that perform the same function, regardless of structure.

Thus, for example, it will be appreciated by those skilled in the art that the block diagrams and flowcharts described hereinabove represent conceptual views of illustrative circuitry and processes embodying the principles of the invention. Similarly, it will be appreciated that any flowcharts, flow diagrams, state transition diagrams, pseudocode, and the like represent various processes which may be substantially represented in computer readable medium and so executed by a computer or processor, whether or not such a computer or processor is explicitly shown.

The functions of the various elements shown in the FIGS., including functional blocks labeled as "processors" may be provided through the use of dedicated hardware as well as hardware capable of executing software in association with appropriate software. When provided by a processor, the functions may be provided by a single dedicated processor, by a single shared processor, or by a plurality of individual processors, some of which may be shared. Moreover, explicit use of the term "processor" or "controller" should not be construed to refer exclusively to hardware capable of executing software, and may implicitly include, without limitation, digital signal processor (DSP) hardware, read-only memory (ROM) for storing software, random access memory (RAM), and non-volatile storage. Other hardware, conventional and/or custom, may also be included. Similarly, any switches shown in the FIGS. are conceptual only. Their function may be carried out through the operation of program logic, through dedicated logic, through the interaction of program control and dedicated logic, or even manually, the particular technique being selectable by the implement as more specifically understood from the context.

In the claims hereof any element expressed as a means for performing a specified function is intended to encompass any way of performing that function including, for example, a) a combination of circuit elements which performs that function or b) software in any form, including, therefore, firmware, microcode or the like, combined with appropriate circuitry for executing that software to perform the function. The invention as defined by such claims resides in the fact that the functionalities provided by the various recited means are combined and brought together in the manner which the claims call for. Applicant thus regards any means which can provide those functionalities as equivalent to those shown hereinabove.

The invention claimed is:

1. A method of processing client requests through at least one redirector to a plurality of servers connected on a communications network to minimize an average delay associated with the client requests, at least some of the client requests being capable of being satisfied by more than one of the servers, the method comprising the steps of:

a) determining an access rate of requests associated with each of a plurality of redirector-logical item pairs;

b) determining a network delay between each of a plurality of clients and the plurality of servers;

c) determining a server delay incurred in processing a client request at each of the plurality of servers;

d) using the determined access rates of requests in step a), the network delays determined in step b) and the server delays determined in step c) as inputs, solving a non-linear program optimization problem to determine a set of weights associated with each of the plurality of redirector-logical item pairs so as to minimize the average delay associated with the client requests; and

e) probabilistically forwarding a client request through the at least one redirector to a server that can satisfy that request using the determined weights associated with the redirector-logical pair item.

2. The method of claim **1** wherein the step d) comprises the step of formulating and solving a minimum cost network flow problem.

3. The method of claim **2** further comprising the step of first determining an initial distribution that maps logical items onto the servers.

4. The method of claim **2** wherein the logical items are a plurality of hot sites.

5. The method of claim **4** wherein the servers are caching servers that each replicate at least one of the hot sites.

6. The method of claim **2** wherein the logical items are groups of clients.

7. The method of claim **6** wherein the groups of clients are determined by their IP addresses.

8. The method of claim **7** wherein the servers are web servers.

9. The method of claim **7** wherein the servers are caching servers.

10. The method of claim **2** wherein the logical items are groups of documents.

11. The method of claim **10** wherein the groups of documents are determined by their origin server IP addresses.

12. The method of claim **10** wherein the at least one redirector is a content-smart switch and the groups of documents are determined by their URLs.

13. The method of claim **11** wherein the servers are web servers.

14. The method of claim **11** wherein the servers are caching servers.

15. The method of claim **2** wherein the at least one redirector is a virtual server for a plurality of web servers.

16. The method of claim 2 wherein steps a) through d) are periodically repeated to determine a new set of weights associated with each redirector-logical item pair to thereafter be used by the at least one redirector.

17. The method of claim 2 wherein steps a) through d) are repeated when a change of an access rate of requests at a redirector is detected.

18. The method of claim 2 wherein steps a) through d) are repeated when a server failure is detected.

19. The method of claim 2 wherein steps a) through d) are repeated when a change in network delay is detected.

20. The method of claim 2 wherein steps a) through d) are repeated when a change in the delay at a server is detected.

21. The method of claim 2 wherein a central management station (CMS) connected on the communications network collects the determined access rates of requests in step a) and the network delay in step b) from the at least one redirector, and the server delay in step c) from the plurality of servers, and then performs step d), the method then further comprising the step of forwarding the determined weights to the at least one redirector.

22. In a system which processes client requests through at least one redirector to a plurality of servers connected on a communications network, at least some of the client requests being capable of being satisfied by more than one of the servers, apparatus for minimizing an average delay associated with the client requests, the apparatus comprising:

means for determining an access rate of requests associated with each of a plurality of redirector-logical item pairs;

means for determining a network delay between each of a plurality of clients and the plurality of servers;

means for determining a server delay incurred in processing a client request at each of the plurality of servers;

means for solving a non-linear programming optimization problem to determine a set of weights associated with each of the plurality of redirector-logical items pairs so as to minimize the average delay of client requests using the determined access rate of requests, the determined network delays, and the determined server delays as inputs to the problem; and

means for probabilistically forwarding a client request through the at least one redirector to a server in the system that can satisfy that request using the determined weights associated with the redirector-logical pair item.

23. The apparatus of claim 22 wherein the means for solving a non-linear optimization problem comprises means for formulating and solving a minimum cost network flow problem.

24. The apparatus of claim 23 further comprising means for determining an initial distribution that maps logical items onto the servers.

25. The apparatus of claim 23 wherein the logical items are a plurality of hot sites.

26. The apparatus of claim 25 wherein the servers are caching servers that each replicate at least one of the hot sites.

27. The apparatus of claim 23 wherein the logical items are groups of clients.

28. The apparatus of claim 27 wherein the groups of clients are determined by their IP addresses.

29. The apparatus of claim 28 wherein the servers are web servers.

30. The apparatus of claim 28 wherein the servers are caching servers.

31. The apparatus of claim 23 wherein the logical items are groups of documents.

32. The apparatus of claim 31 wherein the groups of documents are determined by their origin server IP addresses.

33. The apparatus of claim 31 wherein the at least one redirector is a content-smart switch and the groups of documents are determined by their URLs.

34. The apparatus of claim 32 wherein the servers are web servers.

35. The apparatus of claim 32 wherein the servers are caching servers.

36. The apparatus of claim 23 wherein the at least one redirector is a virtual server for a plurality of web servers.

37. The apparatus of claim 23 wherein the means for solving the non-linear optimization problem periodically determines a new set of weights associated with each redirector-logical item pair to thereafter be used by the at least one redirector.

38. The apparatus of claim 23 wherein the means for solving the non-linear optimization problem determines a new set of weights when a change of an access rate at a redirector is detected.

39. The apparatus of claim 23 wherein the means for solving the non-linear optimization problem determines a new set of weights when a server failure is detected.

40. The apparatus of claim 23 wherein the means for solving the non-linear optimization problem determines a new set of weights when a change in network delay is detected.

41. The apparatus of claim 23 wherein the means for solving the non-linear optimization problem determines a new set of weights when a change in the delay at a server is detected.

42. The apparatus of claim 23 further comprising means for forwarding the determined weights to the at least one redirector.

43. In a system which processes client requests through at least one redirector to a plurality of servers connected on a communications network, at least some of the client requests being capable of being satisfied by more than one of the servers, a method of determining a set of weights with which the at least one redirector will probabilistically forward client requests to the server in the system that can satisfy the requests comprising the steps of:

a) determining an access rate of requests associated with each of a plurality of redirector-logical item pairs;

b) determining a network delay between each of a plurality of clients and the plurality of servers;

c) determining a server delay incurred in processing a client request at each of the plurality of servers; and

d) using the determined access rates of requests in step a), the network delays determined in step b) and the server delays determined in step c) as inputs, solving a non-linear program optimization problem to determine the set of weights associated with each of the plurality of redirector-logical item pairs so as to minimize the average delay associated with the client requests.

44. The method of claim 43 wherein the step d) comprises the step of formulating and solving a minimum cost network flow problem.

45. The method of claim 44 further comprising the step of first determining an initial distribution that maps logical items onto the servers.

46. The method of claim 44 wherein the logical items are a plurality of hot sites.

47. The method of claim 46 wherein the servers are caching servers that each replicate at least one of the hot sites.

48. The method of claim 44 wherein the logical items are groups of clients.

49. The method of claim 48 wherein the groups of clients are determined by their IP addresses.

50. The method of claim 49 wherein the servers are web servers.

51. The method of claim 49 wherein the servers are caching servers.

52. The method of claim 44 wherein the logical items are groups of documents.

53. The method of claim 52 wherein the groups of documents are determined by their origin server IP addresses.

54. The method of claim 52 wherein the at least one redirector is a content-smart switch and the groups of documents are determined by their URLs.

55. The method of claim 53 wherein the servers are web servers.

56. The method of claim 53 wherein the servers are caching servers.

57. The method of claim 44 wherein the at least one redirector is a virtual server for a plurality of web servers.

58. The method of claim 44 wherein steps a) through d) are periodically repeated to determine a new set of weights associated with each redirector-logical item pair to thereafter be used by the at least one redirector.

59. The method of claim 44 wherein steps a) through d) are repeated when a change of an access rate of requests at a redirector is detected.

60. The method of claim 44 wherein steps a) through d) are repeated when a server failure is detected.

61. The method of claim 44 wherein steps a) through d) are repeated when a change in network delay is detected.

62. The method of claim 44 wherein steps a) through d) are repeated when a change in the delay at a server is detected.

63. The method of claim 44 wherein a central management station (CMS) connected on the communications network collects the determined access rates of requests in step a) and the network delay in step b) from the at least one redirector, and the server delay in step c) from the plurality of servers, and then performs step d), the method then further comprising the step of forwarding the determined weights to the at least one redirector.

64. A system for processing client requests to a plurality of servers connected on a communications network, at least some of the client requests being capable of being satisfied by more than one of the servers, the system comprising:

a least one redirector; and

at least one processor performing the steps of:

a) determining an access rate of requests associated with each of a plurality of redirector-logical item pairs;

b) determining a network delay between each of a plurality of clients and the plurality of servers;

c) determining a server delay incurred in processing a client request at each of the plurality of servers; and

d) using the determined access rates of requests in step a), the network delays determined in step b) and the server delays determined in step c) as inputs, solving a non-linear program optimization problem to determine a set of weights associated with each of the plurality of redirector-logical item pairs so as to minimize the average delay associated with the client requests;

the at least one redirector using the determined weights associated with each redirector-logical item pair to probabilistically forward each client request to a server that can satisfy that request.

65. The system of claim 64 wherein the at least one processor performs step d) by formulating and solving a minimum cost network flow problem.

66. The system of claim 65 wherein the at least one processor first performs a step of determining an initial distribution that maps logical items onto the servers.

67. The system of claim 65 wherein the logical items are a plurality of hot sites.

68. The system of claim 67 wherein the servers are caching servers that each replicate at least one of the hot sites.

69. The system of claim 65 wherein the logical items are groups of clients.

70. The system of claim 69 wherein the groups of clients are determined by their IP addresses.

71. The system of claim 70 wherein the servers are web servers.

72. The system of claim 70 wherein the servers are caching servers.

73. The system of claim 65 wherein the logical items are groups of documents.

74. The system of claim 73 wherein the groups of documents are determined by their origin server IP addresses.

75. The apparatus of claim 73 wherein the at least one redirector is a content-smart switch and the groups of documents are determined by their URLs.

76. The system of claim 74 wherein the servers are web servers.

77. The system of claim 74 wherein the servers are caching servers.

78. The system of claim 65 wherein the at least one redirector is a virtual server for a plurality of web servers.

79. The system of claim 65 wherein the at least one processor performs steps a) through d) periodically to determine a new set of weights associated with each redirector-logical item pair to be thereafter used by the at least one redirector.

80. The system of claim 65 wherein the at least one processor repeats steps a) through d) when a change of an access rate of requests at a redirector is detected.

81. The system of claim 65 wherein the at least one processor repeats steps a) through d) when a server failure is detected.

82. The system of claim 65 wherein the at least one processor repeats steps a) thought d) when a change in network delay is detected.

83. The system of claim 65 wherein the at least one processor repeats steps a) through d) when a change in the delay at a server is detected.

* * * * *